Appendix A

portals

```
package com.dcr.dvg.model.portal.directory;
/**
 * @(#)PortalDirectoryModel.java
 * <p>
 * ********************************************************************************
 * <p>
 * The <code>PortalDirectoryModel</code> class that is the model of
 *   a Portal Directory.
 * <p>
 * @author              Edward L. Stull
 *      @version 1.5
 * @since               JDK1.1
 */
//345678901234567890123456789012345678901234567890123456789*12345678901234567890123456789O

import java.awt.Color;
import java.io.File;
import java.util.Date;

import javax.swing.event.TableModelListener;
import javax.swing.table.TableModel;
import javax.swing.tree.TreeNode;

import com.dcr.dvg.model.datasource.directory.DataSourceDirectoryModel;
import com.dcr.dve.model.mdb.IMCTreeTableModel;
import com.dcr.dve.model.mdb.IMDbUserComponent;
import com.dcr.dvg.model.tree.explorer.ExplorerDirectoryModel;
import com.dcr.dvg.model.tree.explorer.TreeDirectoryModel;
import com.dcr.dvg.model.tree.TreeNodeModel;
import com.dcr.dvg.util.throwable.DVException;
import com.dcr.dvg.util.throwable.UnableToSavePortalDirectoryFileException;

public class PortalDirectoryModel
  extends ExplorerDirectoryModel {

        // Names of the columns.
        static protected String[] portalAttributeNames = {"Name", "Modified", "Owner"};

        // Types of the columns.
        static protected Class[] portalAttributeTypes = {String.class, Date.class,
String.class};

        public static boolean TRACE = false;

        protected transient static PortalDirectoryModel sessionModel = null;

/**
 * Constructor.
 */
public PortalDirectoryModel() {

        // temporarily set the root to any empty folder
        this(new PortalFolderModel());

        setRoot((TreeNodeModel)getNewModel().getRoot());

        sessionModel = this;
}


/**
 * Constructor with a root of the specifiec PortalFolderModel.
 * <p>
 * @param folder - PortalFolderModel - the root of the model
 */
protected PortalDirectoryModel(PortalFolderModel folder) {

        super(new PortalDirectoryFolderNodeModel(folder));
}


/**
 * Adds a table model listener.
 * <p>
```

```
 * @param listener - TableModelListener
 */
public void addTableModelListener(TableModelListener listener) {

        // do nothing for now
}


/**
 * Implements IMCTreeTableModel.
 * <p>
 * Gets the column class (indicator).
 * <p>
 * @param column - int - class indicator
 */
public Class getColumnClass(int column) {

        return portalAttributeTypes[column];
}


/**
 * Implements IMCTreeTableModel.
 * <p>
 * Gets the column count.
 * <p>
 * @param columnCount - int
 */
public int getColumnCount() {

        return portalAttributeNames.length;
}


/**
 * Implements IMCTreeTableModel.
 * <p>
 * Gets the column name.
 * <p>
 * @param columnName - String
 */
public String getColumnName(int column) {

        return portalAttributeNames[column];
}


/**
 * Implements javax.swing.table.TableModel.
 * <p>
 * Gets the number of records managed by the data source object. A
 * <B>JTable</B> uses this method to determine how many rows it
 * should create and display.  This method should be quick, as it
 * is call by <B>JTable</B> quite frequently.
 * <p>
 * @return the number or rows in the model
 * @see #getColumnCount
 */
public int getRowCount() {

        return getRowCount((TreeNode)getRoot(), 0);
}

/**
 * Implements javax.swing.table.TableModel.
 * <p>
 * Gets the number of records managed by the data source object. A
 * <B>JTable</B> uses this method to determine how many rows it
 * should create and display.  This method should be quick, as it
 * is call by <B>JTable</B> quite frequently.
 * <p>
 * @return rowCount - int - the number or rows in the model
 * @see #getColumnCount
 */
public int getRowCount(TreeNode treeNode, int startingRowCount) {
```

```
        int rowCount = startingRowCount;

        for (int i = 0; i < ((TreeNode)treeNode).getChildCount(); i++) {
                TreeNode childNode = (TreeNode)getChild(treeNode, i);
                if (! childNode.isLeaf())
                        rowCount += getRowCount(childNode, rowCount);
        }

        return rowCount;
}

/**
 * Gets the shared session model.
 *
 * @return sessionModel - PortalDirectoryModel
 */
public static PortalDirectoryModel getSessionModel() {

        return sessionModel;
}


/**
 * Gets the stock Portal Directory Model.
 * <p>
 * @return stockModel - TreeDirectoryModel
 */
public TreeDirectoryModel getStockModel() {

        traceMessage("Initializing the stock Portal Directory Model");

        PortalFolderModel folder = new PortalFolderModel("Registered Portals", "2001
Jan 01", "DataVantage Global");

        return new PortalDirectoryModel(folder);
}


/**
 * Gets the value at a row and column.
 * NOT IMPLEMENTED
 * <p>
 * @param row - int
 * @param column - int
 * @return value - Object
 */
public Object getValueAt(int row, int column) {

        return null;
}


/**
 * Implements IMCTreeTableModel.
 * <p>
 * Gets the value at a folder and column.
 * <p>
 * @param folder - PortalFolderModel
 * @param column - int
 * @return value - Object
 */
public Object getValueAt(PortalFolderModel folder, int column) {

        Object value = null;
        try {
                switch(column) {
                        case 0: // Name
                                value = folder.getName();
                                break;
                    case 1: // Modified
                                value = folder.getLastUpdatedDate();
                                break;
                        case 2: //Owner
                                value = folder.getOwner();
                                break;
            }
```

```
        } catch (SecurityException se) {
        }

        return value;
}


/**
 * Indirect support for implementation IMCTreeTableModel.
 * <p>
 * Gets the value at a portal and column.
 * <p>
 * @param portal - PortalModel
 * @param column - int
 * @return value - Object
 */
public Object getValueAt(PortalModel portal, int column) {

        Object value = null;
        try {
                switch(column) {
                        case 0: // Name
                                value = portal.getName();
                                break;
                    case 1: // Modified
                                value = portal.getDate();
                                break;
                        case 2: //Owner
                                value = portal.getOwner();
                                break;
            }
        } catch (SecurityException se) {
                value = "ERROR";
        }

        return value;
}


/**
 * Implements IMCTreeTableModel.
 * <p>
 * Gets the value at a node and column.
 * <p>
 * @param node - Object
 * @param column - int
 * @return value - Object
 */
public Object getValueAt(Object node, int column) {

        Object value = null;

        IMDbUserComponent typedNode =
(IMDbUserComponent)((PortalDirectoryNodeModel)node).getDirectoryNodeComponent();
        if (typedNode instanceof PortalModel)
                value = getValueAt((PortalModel)typedNode, column);
        else
                value = getValueAt((PortalFolderModel)typedNode, column);

        return value;
}

/**
 * Implements javax.swing.table.TableModel.
 * <p>
 * Answer if the cell is editable.
 * <p>
 * @param row - int
 * @param column - int
 * @return mode - boolean
 */
public boolean isCellEditable(int row, int column) {

        return false;
}
```

```
/**
 * Implements IMCTreeTableModel.
 * <p>
 * Answer if the cell is editable.
 * <p>
 * @param node - Object
 * @param column - int
 * @return mode - boolean
 */
public boolean isCellEditable(Object node, int column) {

        return false;
}


/**
 * Opens a DirectoryModel from a file.
 * <p>
 * @param file - File
 */
public void openModelIn(File file) {

        try {
                setSessionModel((PortalDirectoryModel)getModelIn(file));
        } catch (Throwable throwable) {
        }

        File dataSourceDirectoryFile
          = new File(file.getName().substring(0, file.getName().lastIndexOf(".ser")) +
"_DS.ser");
        if (file.exists())

        DataSourceDirectoryModel.getSessionModel().openModelIn(dataSourceDirectoryFile)
;
}

/**
 * Implements javax.swing.table.TableModel.
 * <p>
 * Remove a table model listener
 * @param listener - TableModelListener
 */
public void removeTableModelListener(TableModelListener listener) {

        // do nothing
}


/**
 * Implements resetTransientValues().
 *
 * Reset the transient values in the just-loaded connections tree.
 */
public void resetTransientValues() {

        traceMessage(getClass() + "resetTransientValues NOT YET IMPLEMENTED");
        //reset the resource bundle value
}


/**
 * Save (serialize) the Directory Model.
 *
 * Override the super to also save the Data Source Directory.
 * <p>
 * @param file - File
 */
public void saveAs(File file) throws Exception {

        super.saveAs(file);

        //also save the data sources
        DataSourceDirectoryModel.getSessionModel().saveAs(
          new File(file.getName().substring(0, file.getName().lastIndexOf(".ser")) +
"_DS.ser"));
```

```
}

/**
 * Sets a new (i.e., empty) Directory Model.
 */
public void setNewModel() {

        setSessionModel((PortalDirectoryModel)getNewModel());

        int[] childIndices = new int[1];
        childIndices[0] = 1;
        TreeNode newChild = (TreeNode)getSessionModel().getRoot();
        Object[] newChildren = new Object[1];
        newChildren[0] = newChild;
        fireTreeNodesInserted(this, getPathToRoot(newChild), childIndices,
newChildren);

        traceMessage(getClass() + "#setNewModel() Successfully setup a new Directory
Model");
}


/**
 * Sets the shared session model.
 *
 * @param sessionModel - PortalDirectoryModel
 */
public static void setSessionModel(PortalDirectoryModel model) {

        sessionModel = (PortalDirectoryModel)model;
}


/**
 * Implements javax.swing.table.TableModel.
 * NOT IMPLEMENTED
 * <p>
 * @param value - Object
 * @param row - int
 * @param column - int
 */
public void setValueAt(Object value, int row, int column) {

        // do nothing here
}


/**
 * Implements IMCTreeTableModel.
 * NOT IMPLEMENTED
 * <p>
 * @param value - Object
 * @param row - int
 * @param column - int
 */
public void setValueAt(Object value, Object node, int column) {

        // do nothing here
}
}

package com.dcr.dvg.model.portal.directory;
/**
 *      @(#)PortalFolderModel.java
 * <p>
 * ****************************************************************
 * ****************************************************************
 * <p>
 * The <code>PortalFolderModel</code> class that is the model of
 *   a Portal Folder.
 * <p>
 * @author              Edward L. Stull
 * @version 1.6
 * @since               JDK1.1
 */
//345678901234567890123456789012345678901234567890*12345678901234567890123456789
```

```
import java.awt.Image;
import java.util.Date;

import com.dcr.dvg.model.TransientChangeOwnerModel;
import com.dcr.dve.model.mdb.IMDbUserComponent;
import com.dcr.dvg.model.tree.explorer.IExplorerFolderModel;

public class PortalFolderModel
  extends TransientChangeOwnerModel
  implements IMDbUserComponent, IExplorerFolderModel {

        protected String name;
        protected String lastUpdatedDate;     //date of creation, e.g., "1993 Mar 12"
        protected String owner;                                  // e.g., "George Lang"

/**
 * Empty constructor.
 */
public PortalFolderModel() {
}


/**
 * Constructor.
 * <p>
 * @param name - java.lang.String - folder name
 * @param lastUpdatedDate - java.lang.String
 * @param owner java.lang.String
 */
public PortalFolderModel(String name, String lastUpdatedDate, String owner) {

        this.name = name;
        this.lastUpdatedDate = lastUpdatedDate;
        this.owner = owner;
}


/**
 * Creates a new PortalFolderModel which is a copy of this one.
 * @return an identical copy of this PortalFolderModel
 */
public Object clone() {

        PortalFolderModel newPortalFolderModel = new PortalFolderModel();

        //copy only local instance vars
        newPortalFolderModel.setDate(new String(getLastUpdatedDate()));
        newPortalFolderModel.setName(new String(getName()));
        newPortalFolderModel.setOwner(new String(getOwner()));

        // Return the newly created PortalFolderModel copy
        return (Object)newPortalFolderModel;
}


/*
 * Gets the last updated date.
 * <p>
 * @return lastUpdatedDate - String
 */
public String getLastUpdatedDate() {

        return lastUpdatedDate;
}


/*
 * Gets the last updated date.
 * <p>
 * @return date timestamp
 */
public static String getLastUpdatedDateDefault() {

        return new Date().toString();
}
```

```
/**
 * Gets the name of the connection.
 * <p>
 * @return String
 */
public String getName() {

        return name;
}


/**
 * Gets the default name.
 * <p>
 * @return String
 */
public static String getNameDefault() {

        return "<< NAME OF FOLDER >>";
}


/**
 * Gets the owner.
 * <p>
 * @return String
 */
public String getOwner() {

        return owner;
}


/**
 * Gets the default owner.
 * <p>
 * @return String
 */
public static String getOwnerDefault() {

        return "<< OWNER OF FOLDER >>";
}


/*
 * Sets the last updated date.
 * <p>
 * @param date a canonical string representation of the date. The result
 * is of the form <code>"Sat Aug 12 02:30:00 PDT 1995"</code>.
 */
public void setDate(String lastUpdatedDate) {

        this.lastUpdatedDate = lastUpdatedDate;
}


/**
 * Sets the name.
 * <p>
 * @param name String
 */
public void setName(String name) {

        this.name = name;
}


/**
 * Sets the owner.
 * <p>
 * @param String
 */
public void setOwner(String owner) {
```

```
        this.owner = owner;
}
}

package com.dcr.dvg.model.portal.directory;
/**
 *      @(#)PortalFolderModel.java
 * <p>
 * ****************************************************************
 * ****************************************************************
 * <p>
 * The <code>PortalFolderModel</code> class that is the model of
 *   a Portal Folder.
 * <p>
 * @author              Edward L. Stull
 * @version 1.6
 * @since               JDK1.1
 */
//345678901234567890123456789012345678901234567890123456789*12345678901234567890123456789

import java.awt.Image;
import java.util.Date;

import com.dcr.dvg.model.TransientChangeOwnerModel;
import com.dcr.dve.model.mdb.IMDbUserComponent;
import com.dcr.dvg.model.tree.explorer.IExplorerFolderModel;

public class PortalFolderModel
  extends TransientChangeOwnerModel
  implements IMDbUserComponent, IExplorerFolderModel {

        protected String name;
        protected String lastUpdatedDate;     //date of creation, e.g., "1993 Mar 12"
        protected String owner;                                  // e.g., "George Lang"

/**
 * Empty constructor.
 */
public PortalFolderModel() {
}


/**
 * Constructor.
 * <p>
 * @param name - java.lang.String - folder name
 * @param lastUpdatedDate - java.lang.String
 * @param owner java.lang.String
 */
public PortalFolderModel(String name, String lastUpdatedDate, String owner) {

        this.name = name;
        this.lastUpdatedDate = lastUpdatedDate;
        this.owner = owner;
}


/**
 * Creates a new PortalFolderModel which is a copy of this one.
 * @return an identical copy of this PortalFolderModel
 */
public Object clone() {

        PortalFolderModel newPortalFolderModel = new PortalFolderModel();

        //copy only local instance vars
        newPortalFolderModel.setDate(new String(getLastUpdatedDate()));
        newPortalFolderModel.setName(new String(getName()));
        newPortalFolderModel.setOwner(new String(getOwner()));

        // Return the newly created PortalFolderModel copy
        return (Object)newPortalFolderModel;
}


/*
```

```
     * Gets the last updated date.
     * <p>
     * @return lastUpdatedDate - String
     */
    public String getLastUpdatedDate() {

            return lastUpdatedDate;
    }


    /*
     * Gets the last updated date.
     * <p>
     * @return date timestamp
     */
    public static String getLastUpdatedDateDefault() {

            return new Date().toString();
    }


    /**
     * Gets the name of the connection.
     * <p>
     * @return String
     */
    public String getName() {

            return name;
    }


    /**
     * Gets the default name.
     * <p>
     * @return String
     */
    public static String getNameDefault() {

            return "<< NAME OF FOLDER >>";
    }


    /**
     * Gets the owner.
     * <p>
     * @return String
     */
    public String getOwner() {

            return owner;
    }


    /**
     * Gets the default owner.
     * <p>
     * @return String
     */
    public static String getOwnerDefault() {

            return "<< OWNER OF FOLDER >>";
    }


    /*
     * Sets the last updated date.
     * <p>
     * @param date a canonical string representation of the date. The result
     * is of the form <code>"Sat Aug 12 02:30:00 PDT 1995"</code>.
     */
    public void setDate(String lastUpdatedDate) {

            this.lastUpdatedDate = lastUpdatedDate;
    }
```

```
/**
 * Sets the name.
 * <p>
 * @param name String
 */
public void setName(String name) {

        this.name = name;
}


/**
 * Sets the owner.
 * <p>
 * @param String
 */
public void setOwner(String owner) {

        this.owner = owner;
}
}

package com.dcr.dvg.view.controller.portal;
/**
 * @(#)PortalDirectory.java
 * <p>
 * ********************************************************************************
 * ********************************************************************************
 * <p>
 * The <code>PortalDirectory</code> is the view of a Portal Directory Model.
 * <p>
 * @author              Edward L. Stull
 * @version 1.0.10
 * @since               JDK 2
 */
//345678901234567890123456789012345678901234567890123456789*12345678901234567890123456789O

import java.io.File;
import java.util.Enumeration;

import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;
import javax.swing.event.TreeExpansionEvent;
import javax.swing.event.TreeExpansionListener;
import javax.swing.event.TreeModelEvent;
import javax.swing.event.TreeModelListener;
import javax.swing.event.TreeSelectionEvent;
import javax.swing.event.TreeSelectionListener;
import javax.swing.event.TreeWillExpandListener;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.TreePath;

import com.klg.jclass.util.JCListenerList;
import com.klg.jclass.util.swing.DefaultRowSortTableModel;
import com.klg.jclass.util.swing.JCSortableTable;

import com.dcr.dve.model.mdb.MDbNodeTreeData;
import com.dcr.dve.view.vcomponent.vcpanel.VCLayout;

import com.dcr.dvg.model.portal.directory.PortalModel;
import com.dcr.dvg.model.portal.directory.PortalFolderModel;
import com.dcr.dvg.model.portal.directory.PortalDirectoryFolderNodeModel;
import com.dcr.dvg.model.portal.directory.PortalDirectoryModel;
import com.dcr.dvg.model.portal.directory.PortalDirectoryPortalNodeModel;
import com.dcr.dvg.model.tree.explorer.IExplorerFolderModel;
import com.dcr.dvg.util.throwable.DVException;
import com.dcr.dvg.view.component.tree.DVGTree;
import com.dcr.dvg.view.component.treeexplorer.TreeExplorerNodeChildrenTable;
import com.dcr.dvg.view.component.treeexplorer.TreeExplorer;
import com.dcr.dvg.view.controller.directory.ExplorerDirectory;

public class PortalDirectory extends ExplorerDirectory {
```

```
        protected PortalDirectoryFolderNodeModel treeRoot = null;

    /**
     * ooo
     * <p>
     * @param portalDirectoryFolderNode  PortalDirectoryFolderNodeModel
     */
    public PortalDirectory(PortalDirectoryFolderNodeModel treeRoot) {

        super();

        /*DVG deleted*///this.treeRoot = treeRoot; IGNORE THIS, FIX LATER
        PortalDirectoryModel.getSessionModel().openModelIn(new
    File(PortalDirectoryModel.getSessionModel().getDefaultSerializationFileName()));
        updateView();
    }


    /**
     * Adds a listener for data source selection events.
     *
     * @param tsl the DataSourceSelectionListener that will be notified when
     *            a node is selected or deselected (a "negative
     *            selection")
     */
    public void addDataSourceSelectionListener(IPortalDirectorySelectionListener listener)
    {

        listeners = JCListenerList.add(listeners, listener);
    }


    /**
     * Fires a terminal (i.e., Portal) Selection Event.
     *
     * @param tsl the IPortalDirectorySelectionListener that will be notified when
     *            a node is selected or deselected (a "negative selection")
     */
    public void fireTerminalSelectionEvent() {

        PortalDirectorySelectionEvent portalSelectionEvent = new
    PortalDirectorySelectionEvent(this);

        Enumeration e = JCListenerList.elements(listeners);
        for (; e.hasMoreElements(); ) {
            IPortalDirectorySelectionListener listener =
    (IPortalDirectorySelectionListener)e.nextElement();
            listener.portalSelectionChanged(portalSelectionEvent);
        }
    }


    /*
     * Gets the folder that owns the selected Portal.
     * <p>
     * @return folder
     */
    public PortalDirectoryFolderNodeModel getFolderNodeOfSelectedPortal() {

        return (PortalDirectoryFolderNodeModel)getFolderNodeOfSelectedNode();
    }


    /**
     * Gets the Portal Directory Model.
     * <p>
     * @return portalDirectoryModel - PortalDirectoryModel
     */
    public PortalDirectoryModel getPortalDirectoryModel() {

        return (PortalDirectoryModel)treeExplorer.getTreeTableModel();
    }


    /**
     * Gets the selected (host) folder.
```

```
 * <p>
 * @return explorerFolder - IExplorerFolderModel
 */
public IExplorerFolderModel getSelectedFolder() {

	return getSelectedPortalFolder();
}

/**
 * Gets the selected db connection from the connection tree.
 * <p>
 * @param selectedPortal - PortalModel - returns null if no connection is selected
 */
public PortalModel getSelectedPortal()
  throws DVException {

	try {
		TreePath treePath = treeExplorer.getSelectionPath();
		if (treePath != null) {
			Object selectedComponent = treePath.getLastPathComponent();
			if (selectedComponent instanceof PortalDirectoryPortalNodeModel)
				return
(PortalModel)((PortalDirectoryPortalNodeModel)selectedComponent).getPortal();
		}
	} catch (Exception exception) {
	}

	return null;
}


/**
 * Get sthe selected db connection folder from the connection tree.
 * <p>
 * @param portalFolderModel - PortalFolderModel - returns null if no portal is
selected
 */
public PortalFolderModel getSelectedPortalFolder() {

	TreePath treePath = getTree().getSelectionPath();
	if (treePath != null) {
		Object selectedComponent = treePath.getLastPathComponent();
		if (selectedComponent instanceof PortalDirectoryFolderNodeModel)
			return
(PortalFolderModel)((PortalDirectoryFolderNodeModel)selectedComponent).getDirectoryNod
eComponent();
	}

	return null;
}

/**
 * Gets the selected data source node in the data source directory.
 * <p>
 * @param selectedPortalNode - PortalModel - returns null if no data source is
selected
 */
public PortalModel getSelectedPortalNode()
  throws DVException {

	try {
		TreePath treePath = treeExplorer.getSelectionPath();
		if (treePath != null) {
			Object selectedComponent = treePath.getLastPathComponent();
			if (selectedComponent instanceof PortalDirectoryPortalNodeModel)
				return
(PortalModel)((PortalDirectoryPortalNodeModel)selectedComponent).getDirectoryNodeCompo
nent();
		}
	} catch (Exception exception) {
	}

	return null;
}
```

```
/**
 * Gets the selected data source paths.
 * <p>
 * @param selectedDataSourcePaths - TreePath[] - returns null if no data sources
selected
 */
public TreePath[] getSelectedPortalPaths() {

        TreePath[] treePaths = treeExplorer.getSelectionPaths();

        TreePath[] portalPaths = null;
        if (treePaths != null) {
                TreePath[] temporarySelectedPortalPaths = new
TreePath[treePaths.length];
                int portalPathCount = 0;
                for (int i = 0; i< treePaths.length; i++) {
                        if (treePaths[i].getLastPathComponent() instanceof
PortalDirectoryPortalNodeModel) {
                                temporarySelectedPortalPaths[portalPathCount] =
treePaths[i];
                                portalPathCount++;
                        }
                }
                portalPaths = new TreePath[portalPathCount];
                System.arraycopy(temporarySelectedPortalPaths, 0, portalPaths, 0,
portalPathCount);
        }

        return portalPaths;
}

/**
 * Gets the selected data sources from the data source directory view.
 * <p>
 * @param selectedPortals - PortalModel[] - returns null if no ata sources are
selected
 */
public PortalModel[] getSelectedPortals()
  throws DVException {

        TreePath[] treePaths = treeExplorer.getSelectionPaths();

        PortalModel[] portals = null;
        if (treePaths != null) {
                PortalModel[] temporaryPortals = new PortalModel[treePaths.length];
                int portalCount = 0;
                for (int i = 0; i< treePaths.length; i++) {
                        Object selectedComponent = treePaths[i].getLastPathComponent();
                        if (selectedComponent instanceof PortalDirectoryPortalNodeModel)
{
                                temporaryPortals[portalCount] =
(PortalModel)((PortalDirectoryPortalNodeModel)selectedComponent).getPortal();
                                portalCount++;
                        }
                }
                portals = new PortalModel[portalCount];
                System.arraycopy(temporaryPortals, 0, portals, 0, portalCount);
        }

        return portals;
}


/**
 * Gets the TreePath of the selected row inside the directory's table view.
 * <p>
 * @param getSelectionPath - TreePath
 */
public TreePath getSelectionPath() {

        return treeExplorer.getSelectionPath();
}

/**
 * Initializes.
 */
```

```
public void init() {

        super.init();

        JCSortableTable se = (JCSortableTable) treeExplorer.getTable();
        // set keys to define the sort-order when clicking on certain columns
        int keys0 [] = {2, 0};
        se.setKeyColumns(0, keys0);
        int keys1 [] = {2, 1, 0};
        se.setKeyColumns(1, keys1);
        int keys2 [] = {2, 0};
        se.setKeyColumns(2, keys2);

        // assume we have a few rows, and expand
        getTree().expandRow(2);
        getTree().expandRow(1);
        getTree().setSelectionRow(2);
}


/**
 * Removes a listener for data source selection (i.e., TreeSelection) events.
 *
 * @param listener - the DataSourceDataSourceDirectorySelectionListener that will be
notified when
 *             a node is selected or deselected (a "negative
 *             selection")
 */
public void removePortalSelectionListener(PortalDirectorySelectionListener listener) {

        listeners = JCListenerList.remove(listeners, listener);
}


/**
 * Sets the <code>PortalDirectoryModel</code> model.
 */
public void setModel(PortalDirectoryModel model) {

        this.model = model;
}


/**
 * Updates the view.
 */
public void updateView() {

        traceMessage("reloadModel >> reloading datasource model into the datasource
diretory");

        if (getModel() != null) {
                removeAll();
                getModel().removeTreeModelListener(this);
        }

        treeRoot =
(PortalDirectoryFolderNodeModel)PortalDirectoryModel.getSessionModel().getRoot();

        PortalDirectoryModel model = new PortalDirectoryModel();
        PortalDirectoryModel.getSessionModel().setRoot(treeRoot);
        setModel(model);
        // must follow setModel
        model.addTreeModelListener(this);

        init();

        getTree().expandRow(0);
        getTree().addSelectionRow(0);
        fireTerminalSelectionEvent();

        refresh();
}
}

package com.dcr.dvg.view.controller.portal;
```

```
/**
 * @(#)PortalDirectoryController.java
 * <p>
 * ********************************************************************
* ********************************************************************
 * <p>
 * The <code>PortalDirectoryController</code> class must be the superclass of any
 * tree in DataVantage Global.
 * <p>
 * @author              Edward L. Stull
 * @version 1.26
 * @since               JDK 2
 */
//345678901234567890123456789012345678901234567890123456789*12345678901234567890123456789

import java.awt.Cursor;
import java.awt.Dimension;
import java.awt.Image;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.beans.PropertyChangeEvent;
import java.io.File;
import java.io.IOException;
import java.net.URL;
import java.util.Hashtable;

import javax.swing.JLayeredPane;
import javax.swing.text.TextAction;
import javax.swing.tree.TreePath;
import javax.swing.Action;

import com.dcr.dve.model.muser.MUserContext;
import com.dcr.dve.view.vcomponent.VCBorderLayout;
import com.dcr.dve.view.vcomponent.vccontrol.VCControlBar;
import com.dcr.dve.view.vcomponent.VCSplitViewer;
import com.dcr.dve.view.vcomponent.vcdialog.VCOptionDialog;
import com.dcr.dve.view.vcomponent.vcpanel.VCScrollViewer;
import com.dcr.dve.view.vprocess.IVPView;
import com.dcr.dvg.view.component.desktop.DVGDesktop;

import com.dcr.dvg.model.portal.directory.PortalModel;
import com.dcr.dvg.model.portal.directory.PortalFolderModel;
import com.dcr.dvg.model.portal.directory.PortalDirectoryFolderNodeModel;
import com.dcr.dvg.model.portal.directory.PortalDirectoryPortalNodeModel;
import com.dcr.dvg.model.portal.directory.PortalDirectoryModel;
import com.dcr.dvg.model.portal.directory.PortalDirectoryNodeModel;
import com.dcr.dvg.model.tree.ITreeFolderNodeModel;
import com.dcr.dvg.model.tree.explorer.TreeDirectoryModel;
import com.dcr.dvg.util.throwable.DVException;
import com.dcr.dvg.util.throwable.UnableToSavePortalDirectoryFileException;
import com.dcr.dvg.view.component.desktop.DVGDesktop;
import com.dcr.dvg.view.controller.directory.DirectoryController;
import com.dcr.dvg.view.controller.directory.DirectoryControllerAction;
import com.dcr.dvg.view.controller.directory.ExplorerDirectory;
import com.dcr.dvg.view.controller.portal.action.EditPortalFolderAction;
import com.dcr.dvg.view.controller.portal.action.AddPortalFolderAction;
import com.dcr.dvg.view.controller.portal.action.EditPortalAction;
import com.dcr.dvg.view.controller.portal.action.AddPortalAction;
import com.dcr.dvg.view.controller.portal.action.OpenPortalAction;
import com.dcr.dvg.view.controller.portal.action.NewPortalDirectoryAction;
import com.dcr.dvg.view.controller.portal.action.OpenPortalDirecrtoryAction;
import com.dcr.dvg.view.controller.portal.action.SavePortalDirectoryAction;
import com.dcr.dvg.view.controller.portal.action.SaveAsPortalDirectoryAction;
import com.dcr.dvg.view.controller.portal.action.DeletePortalDirectoryElementAction;

import com.dcr.dvg.view.controller.portal.action.CopyPortalDirectoryElementAction;
import com.dcr.dvg.view.controller.portal.action.CutPortalDirectoryElementAction;
import com.dcr.dvg.view.controller.portal.action.DeletePortalDirectoryElementAction;
import com.dcr.dvg.view.controller.portal.action.PastePortalDirectoryElementAction;
import
com.dcr.dvg.view.controller.portal.action.PortalDirectoryControllerFolderAction;
import
com.dcr.dvg.view.controller.portal.action.PortalDirectoryControllerInPlaceAction;

import com.dcr.dvg.view.controller.portal.action.TogglePortalPreviewingAction;
```

```
import com.dcr.dvg.view.controller.portal.action.ToggleShowAsPortalDbResultsAction;

import com.dcr.dvg.view.controller.portal.action.PortalDirectoryControllerHelpAction;
import com.dcr.dvg.view.controller.portal.action.PortalDirectoryExitAction;
import com.dcr.dvg.view.controller.portal.exception.NoPortalSelectedException;
import com.dcr.dvg.view.controller.portal.exception.NoPortalFolderSelectedException;
import com.dcr.dvg.view.controller.portal.exception.UnableToAddPortalException;
import com.dcr.dvg.view.controller.portal.exception.UnableToAddPortalFolderException;
import
com.dcr.dvg.view.controller.portal.exception.UnableToCopyPortalDirectoryElementExcepti
on;
import
com.dcr.dvg.view.controller.portal.exception.UnableToCutPortalDirectoryElementExceptio
n;
import
com.dcr.dvg.view.controller.portal.exception.UnableToDeletePortalDirectoryElementExcep
tion;
import
com.dcr.dvg.view.controller.portal.exception.UnableToNewPortalDirectoryException;
import
com.dcr.dvg.view.controller.portal.exception.UnableToPastePortalDirectoryElementExcept
ion;
import com.dcr.dvg.view.controller.portal.exception.UnableToOpenPortalException;
import
com.dcr.dvg.view.controller.portal.exception.UnableToOpenPortalDirectoryException;
import
com.dcr.dvg.view.controller.portal.exception.UnableToSavePortalDirectoryException;
import
com.dcr.dvg.view.controller.portal.exception.UnableToSaveAsPortalDirectoryException;

import com.dcr.dvg.view.controller.iteration.IterationControllerFrame;
import com.dcr.dvg.view.desktop.DVGDesktopViewer;
import com.dcr.dvg.view.portal.PortalViewer;
import com.dcr.dvg.view.portal.PortalViewerFrame;

public class PortalDirectoryController extends DirectoryController {

        protected PortalDirectoryControllerFrame previewPortalFrame = null;

        protected boolean showAsDbResults = false;
        protected boolean previewingOfPortals = false;

        protected ExistingPortalFolderWizard existingPortalFolderPanel;
        protected NewPortalFolderWizard newPortalFolderPanel;
        protected ExistingPortalWizard existingPortalPanel;
        protected NewPortalWizard newPortalPanel;

/**
 * Constructor.
 */
public PortalDirectoryController() {

        super();
}


/**
 * Constructor based on the desktopView.
 * <p>
 * @param desktopView - JLayeredPane
 */
public PortalDirectoryController(JLayeredPane desktopView) {

        super();

        setDesktopView(desktopView);

        initialize();
}


/**
 * Present a dialog containing a new portal information panel.
 * <p>
 * @param action - PortalDirectoryControllerFolderAction - action to apply
 */
```

```
public void actionAddPortal(PortalDirectoryControllerFolderAction action) {

        applyOperation(action);
}

/**
 * Add a new portal folder.
 * <p>
 * @param action - PortalDirectoryControllerFolderAction - action to apply
 */
public void actionAddPortalFolder(PortalDirectoryControllerFolderAction action) {

        applyOperation(action);
}

/**
 * Copy to the clipboard the selected Portal Directory Node Action.
 * <p>
 * @param action - PortalDirectoryControllerFolderAction - action to apply
 */
public void actionCopyPortalDirectoryNode(PortalDirectoryControllerFolderAction
action) {

        applyOperation(action);
}


/**
 * Cut to the clipboard the selected Portal Directory Node Action.
 * <p>
 * @param action - PortalDirectoryControllerFolderAction - action to apply
 */
public void actionCutPortalDirectoryNode(PortalDirectoryControllerFolderAction action)
{

        applyOperation(action);
}


/**
 * Delete a portal node.
 * <p>
 * @param action - PortalDirectoryControllerFolderAction - action to apply
 */
public void actionDeletePortalDirectoryNode(PortalDirectoryControllerFolderAction
action) {

        applyOperation(action);
}

/**
 * Edit a portal.
 * <p>
 * @param action - PortalDirectoryControllerInPlaceAction - action to apply
 */
public void actionEditPortal(PortalDirectoryControllerInPlaceAction action) {

        applyOperation(action);
}

/**
 * Edit a portal folder.
 * <p>
 * @param action - PortalDirectoryControllerInPlaceAction - action to apply
 */
public void actionEditPortalFolder(PortalDirectoryControllerInPlaceAction action) {

        applyOperation(action);
}

/**
 * Get new portal directory.
 * <p>
 * @param action - PortalDirectoryControllerInPlaceAction - action to apply
 */
public void actionNewPortalDirectory(PortalDirectoryControllerInPlaceAction action) {
```

```
        applyOperation(action);
}

/**
 * Open the portal selected in the portal directory viewer.
 * <p>
 * @param action - PortalDirectoryControllerInPlaceAction - action to apply
 */
public void actionOpenPortal(PortalDirectoryControllerInPlaceAction action) {

        applyOperation(action);
}

/**
 * Open a "saved" portal directory.
 * <p>
 * @param action - PortalDirectoryControllerInPlaceAction - action to apply
 */
public void actionOpenPortalDirectory(PortalDirectoryControllerInPlaceAction action) {

        applyOperation(action);
}

/**
 * Paste Portal Directory Node from the clipboard to the selected Portal Directory
folder Action.
 * <p>
 * @param action - PortalDirectoryControllerFolderAction - action to apply
 */
public void actionPastePortalDirectoryNode(PortalDirectoryControllerFolderAction
action) {

        applyOperation(action);
}


/**
 * Save As the portal directory.
 * <p>
 * @param action - PortalDirectoryControllerInPlaceAction - action to apply
 */
public void actionSaveAsPortalDirectory(PortalDirectoryControllerInPlaceAction action)
{

        applyOperation(action);
}


/**
 * Save the portal directory.
 * <p>
 * @param action - PortalDirectoryControllerInPlaceAction - action to apply
 */
public void actionSavePortalDirectory(PortalDirectoryControllerInPlaceAction action) {

        applyOperation(action);
}


/**
 * Save the portals.
 */
public void actionTogglePreviewingOfPortals() {

        if (! getPreviewingOfPortals()) {
                setPreviewingOfPortals(true);
                showPreviewPortalFrame();
        } else {
                hidePreviewPortalFrame();
                setPreviewingOfPortals(false);
        }
}


/**
```

```
 * Save the portals.
 */
public void actionToggleShowAsDbResults() {

        if (! getShowAsDbResults())
                setShowAsDbResults(true);
        else
                setShowAsDbResults(false);
}


/**
 * Add a new portal folder.
 * <p>
 * @param folderNode - ITreeFolderNodeModel - not currently used
 */
public boolean addFolder(ITreeFolderNodeModel folderNode) {

        boolean completed = true;

        int selection = VCOptionDialog.showOptionDialog(
                this,
        //parentComponent
                newPortalFolderPanel.update(),                  //message - object to
display
                "New Portal Folder Information",        //title - title string for the
dialog
                VCOptionDialog.DEFAULT_OPTION,                  //optionType -
YES_NO_OPTION or YES_NO_CANCEL_OPTION
                VCOptionDialog.QUESTION_MESSAGE,        //messageType - ERROR_MESSAGE,
INFORMATION_MESSAGE, etc
                null,                                                                   //icon -
icon to display in the dialog
                AddElementOptionNames,                          //options - array of
possible choices
                AddElementOptionNames[1]);                      //initialValue - object
that is the default selection

        if(selection == VCOptionDialog.OK_OPTION) {
                setVisible(true);
                newPortalFolderPanel.getEditor().setFolder();


        newPortalFolderPanel.getEditor().getFolder().setOwner(MUserContext.getUserId())
;

                try {
                        getPortalDirectoryModel().insertNode (
                          new
PortalDirectoryFolderNodeModel(newPortalFolderPanel.getEditor().getFolder()),
                            getFolderNodeOfSelectedPortal()
                        );

                        getPortalDirectory().invalidate();
                } catch (Exception exception) {
                        fireDbResultsException(new
UnableToAddPortalFolderException(exception));
                        completed = false;
                }
        } else
                completed = false;

        return completed;
}

/**
 * Present a dialog containing a new portal information panel.
 * <p>
 * @param folderNode - ITreeFolderNodeModel
 */
public boolean addTerminal(ITreeFolderNodeModel folderNode) {

        boolean completed = true;

        int selection = VCOptionDialog.showOptionDialog(
```

```
                this,
        //parentComponent
                newPortalPanel.update(),                        //message - object to
display
                "New Portal Information",                       //title - title string for
the dialog
                VCOptionDialog.DEFAULT_OPTION,                  //optionType -
YES_NO_OPTION or YES_NO_CANCEL_OPTION
                VCOptionDialog.QUESTION_MESSAGE,      //messageType - ERROR_MESSAGE,
INFORMATION_MESSAGE, etc
                null,                                                   //icon -
icon to display in the dialog
                AddElementOptionNames,                          //options - array of
possible choices
                AddElementOptionNames[1]);                      //initialValue - object
that is the default selection

        if(selection == VCOptionDialog.OK_OPTION) {
                setVisible(true);
                newPortalPanel.getEditor().setModel();

                try {
                        getPortalDirectoryModel().insertNode(
                          new
PortalDirectoryPortalNodeModel(newPortalPanel.getEditor().getPortal()),
                            folderNode
                        );
                } catch (Exception exception) {
                        fireDbResultsException(new
UnableToAddPortalException(exception));
                        completed = false;
                }
        } else
                completed = false;

        return completed;
}

/**
 * Clear the results viewer. DO NOTHING HERE
 */
public void clearView() {
}


/**
 * Add a directory element.
 * <p>
 * @param folderNode - ITreeFolderNodeModel - not currently used
 */
public boolean copyNode(ITreeFolderNodeModel folderNode) {

        boolean completed = true;

        try {

        getPortalDirectoryModel().copyToClipBoardNodeOf((PortalDirectoryNodeModel)getPo
rtalDirectory().getSelectedNode());

                getPortalDirectory().invalidate();

                firePostStatus(getResourcesName(), getResources(),
"PortalDirectoryElementCopied");
        } catch (Exception exception) {
                fireDbResultsException(new
UnableToCopyPortalDirectoryElementException(exception));
                completed = false;
        }

        return completed;
}

/**
 * Cut a directory element.
 * <p>
 * @param folderNode - ITreeFolderNodeModel - not currently used
```

```
 */
public boolean cutNode(ITreeFolderNodeModel folderNode) {

        boolean completed = true;

        try {

        getPortalDirectoryModel().cutToClipBoardNodeOf((PortalDirectoryNodeModel)getPor
talDirectory().getSelectedNode());

                getPortalDirectory().invalidate();

                //will reset clipboard ==> updateView();

                firePostStatus(getResourcesName(), getResources(),
"PortalDirectoryElementCut");
        } catch (Exception exception) {
                fireDbResultsException(new
UnableToCutPortalDirectoryElementException(exception));
                completed = false;
        }

        return completed;
}

/**
 * Delete a directory element.
 * <p>
 * @param folderNode - ITreeFolderNodeModel - not currently used
 */
public boolean deleteNode(ITreeFolderNodeModel folderNode) {

        boolean completed = true;

        try {

        getPortalDirectoryModel().removeNode((PortalDirectoryNodeModel)getPortalDirecto
ry().getSelectedNode());

                getPortalDirectory().invalidate();

                updateView();
        } catch (Exception exception) {
                fireDbResultsException(new
UnableToDeletePortalDirectoryElementException(exception));
                completed = false;
        }

        return completed;
}

/**
 * Disable the frame being used for previewing the database portal.
 */
public void diablePreviewFrame() {

        setPreviewPortalFrame(null);
}


/**
 * Edit a portal folder.
 * <p>
 * @param completed - boolean - true if successful
 */
public boolean editFolder() {

        boolean completed = true;

        Object node = getPortalDirectory().getSelectedPortalFolder();

        if (node instanceof PortalFolderModel)
                editPortalFolder((PortalFolderModel)node);
        else {
                fireDbResultsException(new NoPortalFolderSelectedException());
                completed = false;
```

```
        }

        return completed;
    }

    /**
     * Edit a portal.
     * <p>
     * @param completed - boolean - true if successful
     */
    public boolean editPortal() {

        boolean completed = true;

        Object node = getSelectedPortal();

        if (node instanceof PortalModel)
                editPortal((PortalModel)node);
        else {
                fireDbResultsException(new NoPortalSelectedException());
                completed = false;
        }

        return completed;
    }

    /**
     * Bring up a dialog containing a new portal information panel.
     * <p>
     * @param portal - PortalModel
     */
    public void editPortal(PortalModel portal) {

        notifyOperationBegun("Edit Portal");

        int selection = VCOptionDialog.showOptionDialog(
                this,
        //parentComponent
                existingPortalPanel.update(portal),   //message - object to display
                "Edit Portal Information",                    //title - title string for
the dialog
                VCOptionDialog.DEFAULT_OPTION,                //optionType -
YES_NO_OPTION or YES_NO_CANCEL_OPTION
                VCOptionDialog.QUESTION_MESSAGE,      //messageType - ERROR_MESSAGE,
INFORMATION_MESSAGE, etc
                null,                                                         //icon -
icon to display in the dialog
                EditElementOptionNames,                                  //options - array
of possible choices
                EditElementOptionNames[1]);                   //initialValue - object
that is the default selection

        if(selection == VCOptionDialog.OK_OPTION) {
                setVisible(true);
                existingPortalPanel.getEditor().setModel();

                notifyOperationFinished("Edit Portal");
        } else {
                notifyOperationCanceled("Edit Portal");

                if(! this.isVisible())
                        //DV was shut down before completion of this action
                        System.exit(0);
        }
    }

    /**
     * Bring up a dialog containing a new portal information panel.
     * <p>
     * @param portalFolder - PortalFolderModel
     */
    public void editPortalFolder(PortalFolderModel portalFolder) {

        notifyOperationBegun("Edit Portal Folder");

        int selection = VCOptionDialog.showOptionDialog(
```

```
                this,
        //parentComponent
                existingPortalFolderPanel.update(portalFolder), //message - object to
display
                "Edit Portal Information",                      //title - title string for
the dialog
                VCOptionDialog.DEFAULT_OPTION,                  //optionType -
YES_NO_OPTION or YES_NO_CANCEL_OPTION
                VCOptionDialog.QUESTION_MESSAGE,      //messageType - ERROR_MESSAGE,
INFORMATION_MESSAGE, etc
                null,                                                   //icon -
icon to display in the dialog
                EditElementOptionNames,                                 //options - array
of possible choices
                EditElementOptionNames[1]);                     //initialValue - object
that is the default selection

        if(selection == VCOptionDialog.OK_OPTION) {
                setVisible(true);
                existingPortalFolderPanel.getEditor().setFolder();

                notifyOperationFinished("Edit Portal Folder");
        } else {
                notifyOperationCanceled("Edit Portal Folder");

                if(! this.isVisible())
                        //DV was shut down before completion of this action
                        System.exit(0);
        }
}

/**
 * Gets the list of actions supported by this launcher.
 * @return the list of actions supported by the embedded JTextComponent
 *                              augmented with the actions defined locally.
 */
public Action[] getActions() {

        Action[] defaultActions = {
            new EditPortalFolderAction(this),
            new AddPortalFolderAction(this),

            new EditPortalAction(this),
            new AddPortalAction(this),

            new OpenPortalAction(this),

            new NewPortalDirectoryAction(this),
            new OpenPortalDirecrtoryAction(this),
            new SavePortalDirectoryAction(this),
            new SaveAsPortalDirectoryAction(this),

                new CopyPortalDirectoryElementAction(this),
                new CutPortalDirectoryElementAction(this),
                new PastePortalDirectoryElementAction(this),
                new DeletePortalDirectoryElementAction(this),

            new TogglePortalPreviewingAction(this),
            new ToggleShowAsPortalDbResultsAction(this),

            new PortalDirectoryControllerHelpAction(this),

            new PortalDirectoryExitAction(this),
        };

        return TextAction.augmentList(super.getActions(), defaultActions);
}

/**
 * Gets the Explorer Directory view.
 * <p>
 * @return explorerDirectory  ExplorerDirectory
 */
public ExplorerDirectory getExplorerDirectory() {

        return getPortalDirectory();
```

```
}


/*
 * Gets the folder node that owns this Portal.
 * <p>
 * @return folderModel - PortalDirectoryFolderNodeModel
 */
public PortalDirectoryFolderNodeModel getFolderNodeOfSelectedPortal() {

        return
(PortalDirectoryFolderNodeModel)getPortalDirectory().getFolderNodeOfSelectedPortal();
}


/**
 * Gets the type name of the node (e.g., Portal, Data Source) managed by the
controller.
 * <p>
 * @return nodeTypeName  String
 */
public String getNodeTypeName() {

        return "Portal";
}

/**
 * Gets the exception for "no folder selected".
 * <p>
 * @return exception - DVException
 */
public DVException getNoFolderSelectedException() {

        return new NoPortalFolderSelectedException();
}


/**
 * Gets the portal diretory.
 * <p>
 * @return portalDirectory - PortalDirectory
 */
public PortalDirectory getPortalDirectory() {

        return ((PortalDirectoryView)getView()).getPortalDirectory();
}


/**
 * Gets the portal tree.
 * <p>
 * @return portalTree PortalDirectoryModel
 */
public PortalDirectoryModel getPortalDirectoryModel() {

        return (PortalDirectoryModel)getPortalDirectory().getModel();
}


/**
 * Answer if portals are to be previewed.
 * <p>
 * @return previewingOfPortals - boolean
 */
public boolean getPreviewingOfPortals() {

        return previewingOfPortals;
}


/**
 * Gets the preview portal frame.
 * <p>
 * @param previewPortalFrame - PortalDirectoryControllerFrame
 */
public PortalDirectoryControllerFrame getPreviewPortalFrame() {
```

```
        if (previewPortalFrame == null)
                setPreviewPortalFrame(queNewDbResultsViewerFrame());

        return previewPortalFrame;
}


/**
 * Gets the selected portal from the portals tree.
 * <p>
 * @param selectedPortal - PortalModel - returns null if no portal is selected
 */
public PortalModel getSelectedPortal() {

        Object node =  null;

        try {
                node = getPortalDirectory().getSelectedPortal();
        } catch (Exception exception) {
        }

        return (PortalModel)node;
}


/**
 * Answer if the portal is to be shown as DB Results.
 * <p>
 * @return showAsDbResults - boolean
 */
public boolean getShowAsDbResults() {

        return showAsDbResults;
}


/**
 * Gets the Tree Directory Model.
 * <p>
 * @return portalDirectoryModel - TreeDirectoryModel
 */
public TreeDirectoryModel getTreeDirectoryModel() {

        return getPortalDirectoryModel();
}


/**
 * Hide the preview portal frame.
 */
public void hidePreviewPortalFrame() {

        removePropertyChangeListener(getPreviewPortalFrame());

        getPreviewPortalFrame().setVisible(false);
}


/**
 * Creates the command line content area.
 * <p>
 * @return success boolean
 */
public boolean initialize() {

        initialize("PortalDirectoryController", (IVPView)new PortalDirectoryView());

        getPortalDirectory().addTreeSelectionListener(new
PortalDirectorySelectionListener(this));

        getPortalDirectory().expandRow(1);

        existingPortalFolderPanel = new ExistingPortalFolderWizard(this);
        newPortalFolderPanel = new NewPortalFolderWizard(this);
        existingPortalPanel = new ExistingPortalWizard(this);
```

```
        newPortalPanel = new NewPortalWizard(this);

        //DONE LATER setPreviewPortalFrame(queNewDbResultsViewerFrame());

        return true;
}


/**
 * Launches the help facility for this viewer's context.
 */
public void launchViewerContextHelp() {

        launchViewerContextHelpUsing("portalDirectoryController");
}


/**
 * Sets a new portal directory.
 * <p>
 * @param completed - boolean - true if successful
 */
public boolean newPortalDirectory() {

        boolean completed = true;

        try {
                PortalDirectoryModel.getSessionModel().setNewModel();
        } catch (Exception exception) {
                fireDbResultsException(new
UnableToNewPortalDirectoryException(exception));
                completed = false;
        }

        return completed;
}

/**
 * Open a "saved" portal directory.
 * <p>
 * @param completed - boolean - true if successful
 */
public boolean openDirectory() {

        boolean completed = true;

        File file = getFileToOpen();

        try {
                if (file != null) {
                        PortalDirectoryModel.getSessionModel().openModelIn(file);

                        updateView();
                }
        } catch (Exception exception) {
                fireDbResultsException(new
UnableToOpenPortalDirectoryException(exception));
                completed = false;
        }

        return completed;
}

/**
 * Opens the portal selected in the portal directory viewer.
 * <p>
 * @param completed - boolean - true if successful
 */
public boolean openPortal() {

        boolean completed = true;

        Object node =  null;

        try {
                node = getPortalDirectory().getSelectedPortal();
```

```
        } catch (Exception exception) {
        }

        traceMessage("\n" + getClass() + "  >>  openPortal() on node:" + node);

        PortalViewer portalViewer = null;

        if (node instanceof PortalModel)
                portalViewer = openPortal((PortalModel)node);
        else {
                clearView();
                fireDbResultsException(new NoPortalSelectedException());
                completed = false;
        }

        return completed;
}

/**
 * Opens the specified portal.
 * <p>
 * @param portalModel - PortalModel - portal to open
 */
public PortalViewer openPortal(PortalModel portalModel) {

        firePostStatus(
          getResourcesName(),
          getResources(),
          "OpeningPortal",
          "  \"" + portalModel.getName() + "\"."
        );

        PortalViewerFrame portalFrame = getDesktopViewer().getNewPortalFrame();
//      portalFrame.fireExecuteCommands(/*commandText*/"");
        portalFrame.refresh();

        portalFrame.viewPortal(portalModel);
        portalFrame.setVisible(true);

        firePostStatus(
          "\"" + portalModel.getName() + "\" ",
          getResourcesName(),
          getResources(),
          "PortalOpened"
        );

        return portalFrame.getPortalViewer();
}

/**
 * Pastes a Directory Node.
 * <p>
 * @param completed - boolean - true if successful
 */
public boolean pasteNode(ITreeFolderNodeModel folderNode) {

        boolean completed = true;

        try {
                getPortalDirectoryModel()

.pasteClipBoardNodeTo((PortalDirectoryFolderNodeModel)getPortalDirectory().getFolderNo
deOfSelectedPortal());

                getPortalDirectory().invalidate();
        } catch (Exception exception) {
                fireDbResultsException(new
UnableToPastePortalDirectoryElementException(exception));
                completed = false;
        }

        return completed;
}

/**
 * Que a new DbResultsViewer frame that will be added to the desktop later.
```

```
     * <p>
     * @param resultsViewerFrame PortalDirectoryControllerFrame
     */
    public PortalDirectoryControllerFrame queNewDbResultsViewerFrame() {

            // frame is added to the desktop later
            PortalDirectoryControllerFrame resultsViewerFrame = new
    PortalDirectoryControllerFrame(getDesktopView(), getNewFrameDimension());

            ((DVGDesktop)getDesktopView()).setNextDbResultsFrameCount();
            resultsViewerFrame.setFrameId(((DVGDesktop)getDesktopView()).getDbResultsFrameC
    ount());
            addPropertyChangeListener(resultsViewerFrame);
            resultsViewerFrame.addPropertyChangeListener(getIterationControllerFrame());

            return resultsViewerFrame;
    }

    /**
     * Save As the portal directory.
     * <p>
     * @param completed - boolean - true if successful
     */
    public boolean saveAsDirectory() {

            boolean completed = true;

            File file = getFileToSave();

            try {
                    if (file != null)
                            PortalDirectoryModel.getSessionModel().saveAs(file);
            } catch (Exception exception) {
                    fireDbResultsException(new
    UnableToSaveAsPortalDirectoryException(exception));
                    completed = false;
            }

            return completed;
    }


    /**
     * Saves the portal directory.
     * <p>
     * @param completed - boolean - true if successful
     */
    public boolean saveDirectory() {

            boolean completed = true;

            try {
                    PortalDirectoryModel.getSessionModel().save();
            } catch (Exception exception) {
                    fireDbResultsException(new
    UnableToSavePortalDirectoryException(exception));
                    completed = false;
            }

            return completed;
    }

    /**
     * Sets if portals are to be previewed.
     * <p>
     * @param mode - boolean
     */
    public void setPreviewingOfPortals(boolean mode) {

            previewingOfPortals = mode;
    }

    /**
     * Sets the frame to be used for previewing the portal.
     * <p>
     * @param previewPortalFrame - PortalDirectoryControllerFrame
```

```
 */
public void setPreviewPortalFrame(PortalDirectoryControllerFrame previewPortalFrame) {

        if (previewPortalFrame != null)
                removePropertyChangeListener(previewPortalFrame);

        this.previewPortalFrame = previewPortalFrame;
        // PropertyChangeListener added upon showing of preview
}


/**
 * Sets if the portal is to be shown as DB Results.
 * <p>
 * @param mode - boolean
 */
public void setShowAsDbResults(boolean mode) {

        showAsDbResults = mode;
}


/**
 * Show the preview portal frame.
 */
public void showPreviewPortalFrame() {

        addPropertyChangeListener(getPreviewPortalFrame());

        getPreviewPortalFrame().setVisible(true);
}


/**
 * Update the portal directory view.
 */
public void updateView() {

        ((PortalDirectoryView)getView()).getPortalDirectory().updateView();
}

/**
 * Previews the database of the database portal.
 * <p>
 * @param portal - PortalModel - the database portal to preview
 */
public void viewDataBasePortal(PortalModel portal) {

        traceMessage(getClass() + "  >>  viewDataBasePortal");

        fireClearDesktopStatus();

        if (getPreviewingOfPortals()) {
                if (! getShowAsDbResults()) {
                        // just reuse exiting preview frame and preview listener
                        firePropertyChange("Portal Available", null, portal);
                } else {
                        // do not reuse preview frame plus getPreviewingOfPortals()
state does not matter
                        diablePreviewFrame();

                        queNewDbResultsViewerFrame();

                        firePropertyChange("Portal Available", null, portal);
                }
        }
}
}

/**
 * @(#)PortalDirectoryView.java
 * <p>
 * ********************************************************************************
 * ********************************************************************************
 * <p>
 * The <code>PortalDirectoryView</code> is a parent view of
```

```
 *   a <code>Portal Directory<code>.
 * <p>
 * @author              Edward L. Stull
 * @version 1.6
 * @since               JDK 2
 */
//345678901234567890123456789012345678901234567890*12345678901234567890123456789

import javax.swing.SwingConstants;

import com.dcr.dve.view.vcomponent.VCBorderLayout;
import com.dcr.dve.view.vcomponent.vcpanel.VCLayout;
import com.dcr.dve.view.vprocess.IVPView;

import com.dcr.dvg.model.portal.directory.PortalDirectoryFolderNodeModel;
import com.dcr.dvg.model.portal.directory.PortalDirectoryModel;
import com.dcr.dvg.view.component.panel.DVGPanel;

public class PortalDirectoryView
  extends DVGPanel
  implements IVPView {

        protected PortalDirectory portalDirectory;
}


package com.dcr.dvg.view.controller.portal;
/**
 * @(#)PortalEditor.java
 * <p>
 * ********************************************************************************
 * ********************************************************************************
 * <p>
 * The <code>PortalEditor</code> class is the editor view of a Portal Model.
 * <p>
 * @author              Edward L. Stull
 * @version 1.9
 * @since               JDK1.1
 */
//345678901234567890123456789012345678901234567890*12345678901234567890123456789

import java.awt.Image;
import java.util.Date;
import java.util.Enumeration;
import java.util.Vector;

import com.klg.jclass.util.swing.JCAlignLayout;

import com.dcr.dve.model.mdb.MDbDrivers;
import com.dcr.dve.view.vcomponent.VCBoxLayout;
import com.dcr.dve.view.vcomponent.VCColor;
import com.dcr.dve.view.vcomponent.VCGridLayout;
import com.dcr.dve.view.vcomponent.vcgraphic.VCGraphic;
import com.dcr.dve.view.vcomponent.vcwizard.VCWizardStep;
import com.dcr.dvg.view.vmodel.VMPassword;
import com.dcr.dve.view.vprocess.VPWizardComboBox;
import com.dcr.dve.view.vprocess.VPWizardField;
import com.dcr.dve.view.vprocess.VPWizardGroupBox;
import com.dcr.dve.view.vprocess.VPWizardLabel;
import com.dcr.dve.view.vprocess.VPWizardReadOnlyField;

import com.dcr.dvg.model.portal.directory.PortalModel;
import com.dcr.dvg.model.portal.directory.PortalDirectoryFolderNodeModel;
import com.dcr.dvg.view.component.panel.DVGPanel;

public class PortalEditor extends DVGPanel {

        protected PortalDirectoryController controller = null;
        protected PortalModel portal = null;

        protected VPWizardField hostFolderNameField;
        protected VPWizardField portalNameField;
        protected VPWizardField lastUpdatedDateField;
        protected VPWizardField portalOwnerNameField;
        protected VPWizardField userIdField;
        protected VMPassword passwordField;
```

```
        protected VPWizardField serverField;
        protected VPWizardComboBox driverField;
        protected VPWizardComboBox accessModeField;

        protected VPWizardField tableQueryField;
        protected VPWizardField metaQueryField;
        protected VPWizardField chartQueryField;
        protected VPWizardField plexusQueryField;
        protected VPWizardField testSetSamplingQueryField;

/**
 * Constructor based on the host view.
 * <p>
 * @param controller - PortalDirectoryController
 */
public PortalEditor(PortalDirectoryController controller) {

        this.controller = controller;

        initialize();
}


/**
 * Gets the controller.
 * <p>
 * @return controller - PortalDirectoryController
 */
public PortalDirectoryController getController() {

        return controller;
}


/*
 * Gets the folder that owns this portal.
 * <p>
 * @return folder - PortalDirectoryFolderNodeModel
 */
public PortalDirectoryFolderNodeModel getFolder() {

        return getController().getFolderNodeOfSelectedPortal();
}

/*
 * Gets the portal being edited.
 * <p>
 * @return portal - PortalModel
 */
public PortalModel getPortal() {

        return portal;
}


/**
 * Initializes.
 */
public void initialize() {

        setLayout(new JCAlignLayout(2, 5, 5));

        add(new VPWizardLabel("Host Folder:"));
        add(hostFolderNameField = new
VPWizardReadOnlyField(PortalModel.getOwnerDefault()));

        add(new VPWizardLabel("Name:"));
        add(portalNameField = new VPWizardField(PortalModel.getNameDefault()));

        add(new VPWizardLabel("Last Updated Date:"));
        add(lastUpdatedDateField = new VPWizardField(PortalModel.getDateDefault()));

        add(new VPWizardLabel("Owner:"));
        add(portalOwnerNameField = new VPWizardField(PortalModel.getOwnerDefault()));
```

```
        add(new VPWizardLabel("User Id:"));
        add(userIdField = new VPWizardField(PortalModel.getUserIdDefault()));

        add(new VPWizardLabel("Password:"));
        add(passwordField = new VMPassword(PortalModel.getUserPasswordDefault()));
}


/*
 * Sets the portal's model.
 * <p>
 * @return portal - PortalModel
 */
public PortalModel setModel() {

        PortalModel portal = getPortal();

        portal.setDate(new Date().toString());
        portal.setName(portalNameField.getText());
        portal.setOwner(portalOwnerNameField.getText());

        return portal;
}


/**
 * Updates the editor with the default portal state.
 * <p>
 * @return portalEditor - PortalEditor - editor used to return an updated object to
the dialog
 */
public PortalEditor update() {

        this.portal = new PortalModel();

        hostFolderNameField.setTemplateText(getFolder().getName());
        portalNameField.setTemplateText(PortalModel.getNameDefault());
        lastUpdatedDateField.setTemplateText(PortalModel.getDateDefault());
        portalOwnerNameField.setTemplateText(PortalModel.getOwnerDefault());
        userIdField.setTemplateText(PortalModel.getUserIdDefault());
        passwordField.setTemplateText(PortalModel.getUserPasswordDefault());

        return this;
}

/**
 * Updates the editor with the portal state.
 * <p>
 * @return portalEditor - PortalEditor - editor used to return an updated object to
the dialog
 */
public PortalEditor update(PortalModel portalToBeEdited) {

        this.portal = portalToBeEdited;

        hostFolderNameField.setTemplateText(getFolder().getName());
        lastUpdatedDateField.setTemplateText(portal.getDate());
        portalNameField.setTemplateText(portal.getName());
        portalOwnerNameField.setTemplateText(portal.getOwner());

        return this;
}
}

package com.dcr.dvg.view.controller.portal;
/**
 * @(#)PortalFolderEditor.java
 * <p>
 * ********************************************************************************
 * ********************************************************************************
 * <p>
 * The <code>PortalFolderEditor</code> class is the editor view of
 *   a Portal Folder Model.
 * <p>
 * @author              Edward L. Stull
 *      @version 1.8
```

```
 * @since                JDK1.1
 */
//3456789012345678901234567890123456789012345678901*12345678901234567890123456789

import java.awt.Image;
import java.util.Date;

import com.klg.jclass.util.swing.JCAlignLayout;

import com.dcr.dve.view.vcomponent.VCBoxLayout;
import com.dcr.dve.view.vcomponent.VCColor;
import com.dcr.dve.view.vcomponent.VCGridLayout;
import com.dcr.dve.view.vcomponent.vcgraphic.VCGraphic;
import com.dcr.dve.view.vcomponent.vcwizard.VCWizardStep;
import com.dcr.dve.view.vprocess.VPWizardField;
import com.dcr.dve.view.vprocess.VPWizardLabel;
import com.dcr.dve.view.vprocess.VPWizardReadOnlyField;

import com.dcr.dvg.model.portal.directory.PortalFolderModel;
import com.dcr.dvg.model.portal.directory.PortalDirectoryFolderNodeModel;
import com.dcr.dvg.view.component.panel.DVGPanel;

public class PortalFolderEditor extends DVGPanel {

        protected PortalDirectoryController controller = null;
        protected PortalFolderModel folder = null;

        protected VPWizardField hostFolderNameField;
        protected VPWizardField portalNameField;
        protected VPWizardField lastUpdatedDateField;
        protected VPWizardField portalOwnerNameField;

/**
 * Constructor based on the host view.
 * <p>
 * @param controller - PortalDirectoryController
 */
public PortalFolderEditor(PortalDirectoryController controller) {

        this.controller = controller;

        initialize();
}


/**
 * Gets the controller.
 * <p>
 * @return controller - PortalDirectoryController
 */
public PortalDirectoryController getController() {

        return controller;
}


/*
 * Gets the folder being edited.
 * <p>
 * @return folder - PortalFolderModel
 */
public PortalFolderModel getFolder() {

        return folder;
}

/*
 * Gets the folder that owns this portal.
 * <p>
 * @return folder - PortalFolderModel
 */
public PortalFolderModel getHostFolder() {

        return
((PortalDirectoryFolderNodeModel)getHostFolderTreeNode()).getPortalFolder();
}
```

```
/*
 * Gets the folder directory node that owns this portal.
 * <p>
 * @return folder - PortalDirectoryFolderNodeModel
 */
public PortalDirectoryFolderNodeModel getHostFolderTreeNode() {

        return getController().getFolderNodeOfSelectedPortal();
}

/*
 * Gets the folder directory node that owns this portal.
 * <p>
 * @return folder - PortalDirectoryFolderNodeModel
 */
public PortalDirectoryFolderNodeModel getTreeNode() {

        return getController().getFolderNodeOfSelectedPortal();
}

/**
 * Initializes.
 */
public void initialize() {

        setLayout(new JCAlignLayout(2, 5, 5));

        add(new VPWizardLabel("Host Folder:"));
        add(hostFolderNameField = new
VPWizardReadOnlyField(PortalFolderModel.getOwnerDefault()));

        add(new VPWizardLabel("Name:"));
        add(portalNameField = new VPWizardField(PortalFolderModel.getNameDefault()));

        add(new VPWizardLabel("Last Updated Date:"));
        add(lastUpdatedDateField = new
VPWizardField(PortalFolderModel.getLastUpdatedDateDefault()));

        add(new VPWizardLabel("Owner:"));
        add(portalOwnerNameField = new
VPWizardField(PortalFolderModel.getOwnerDefault()));
}


/*
 * Sets the state of the portal folder model.
 * <p>
 * @return folder - PortalFolderModel
 */
public PortalFolderModel setFolder() {

        getFolder().setDate(new Date().toString());
        getFolder().setName(portalNameField.getText());
        getFolder().setOwner(portalOwnerNameField.getText());

        return getFolder();
}


/**
 * Updates the editor with the default portal state.
 * <p>
 * @return portalEditor - PortalEditor - editor used to return an updated object to
the dialog */
public PortalFolderEditor update() {

        this.folder = new PortalFolderModel();

        hostFolderNameField.setTemplateText(getHostFolder().getName());
        portalNameField.setTemplateText(PortalFolderModel.getNameDefault());
        lastUpdatedDateField.setTemplateText(PortalFolderModel.getLastUpdatedDateDefaul
t());
        portalOwnerNameField.setTemplateText(PortalFolderModel.getOwnerDefault());

        return this;
```

```
}

/**
 * Updates the editor with the portal state.
 * <p>
 * @return portalEditor - PortalEditor - editor used to return an updated object to
the dialog
 */
public PortalFolderEditor update(PortalFolderModel folderToBeEdited) {

        this.folder = folderToBeEdited;

        hostFolderNameField.setTemplateText(getHostFolder().getName());
        lastUpdatedDateField.setTemplateText(folderToBeEdited.getLastUpdatedDate());
        portalNameField.setTemplateText(folderToBeEdited.getName());
        portalOwnerNameField.setTemplateText(folderToBeEdited.getOwner());

        return this;
}
}

package com.dcr.dvg.view.portal;
/**
 * @(#)PortalViewer.java
 * <p>
 * ********************************************************************************
 * ********************************************************************************
 * <p>
 * The <code>PortalViewer</code> displays and manages the viewers of a portal.
 * <p>
 * @author              Edward L. Stull
 * @version 2.133
 * @since               JDK 2
 */
//3456789012345678901234567890123456789012345678901234567890*12345678901234567890123456789

import java.awt.Cursor;
import java.awt.FileDialog;
import java.awt.Frame;
import java.beans.PropertyChangeEvent;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.util.Enumeration;
import java.util.Vector;

import javax.swing.Action;
import javax.swing.JInternalFrame;
import javax.swing.JLayeredPane;
import javax.swing.event.InternalFrameEvent;
import javax.swing.event.InternalFrameListener;
import javax.swing.text.TextAction;
import javax.swing.tree.TreeModel;

import com.dcr.dve.model.mdb.MDbJCDataEvent;
import com.dcr.dve.model.mdb.MDbDataModelListener;
import com.dcr.dve.model.mdb.MDbMetaDataModel;
import com.dcr.dve.model.mdb.MDbNodeProperties;
import com.dcr.dve.model.mdb.MDbNodeTreeData;
import com.dcr.dve.model.mdb.MDbTreeProperties;
import com.dcr.dve.view.vcomponent.VCBorderLayout;
import com.dcr.dve.view.vcomponent.vccontrol.VCControlBar;
import com.dcr.dve.view.vcomponent.vclabel.VCLabel;
import com.dcr.dve.view.vcomponent.vctext.VCRichTextParagraph;
import com.dcr.dve.view.vcomponent.vctablerecord.VCTrRecord;
import com.dcr.dve.view.vcomponent.vcplexus.VCPlexus;
import com.dcr.dve.view.vprocess.vphigrid.VPHDesktopHiGridViewerFrame;
import com.dcr.dve.view.vprocess.vphigrid.VPVHiGridExternalDS;

import com.dcr.dvg.model.MessageKit;
import com.dcr.dvg.model.TransientChangeOwnerModel;
import com.dcr.dvg.model.portal.directory.PortalModel;
import com.dcr.dvg.model.session.SessionModel;
import com.dcr.dvg.model.viewer.ViewerModel;
import com.dcr.dvg.model.viewer.DataSourceViewerModel;
```

```
import com.dcr.dvg.model.viewer.MViewerChart;
import com.dcr.dvg.model.viewer.MViewerCrystalReports;
import com.dcr.dvg.model.viewer.MViewerHiGrid;
import com.dcr.dvg.model.viewer.MViewerPlexus;
import com.dcr.dvg.model.viewer.MViewerRecord;
import com.dcr.dvg.model.viewer.MViewerSql;
import com.dcr.dvg.model.viewer.MViewerMeta;
import com.dcr.dvg.view.component.desktop.DVGDesktop;
import com.dcr.dvg.view.component.desktop.DVGInternalFrame;
import com.dcr.dvg.view.controller.datasource.DataSourceDirectory;
import com.dcr.dvg.view.controller.datasource.DataSourceDirectoryControllerFrame;
import com.dcr.dvg.view.controller.transcript.TranscriptControllerFrame;
import com.dcr.dvg.view.desktop.DVGDesktopViewer;
import com.dcr.dvg.view.dsprojection.InternalResultsViewer;
import com.dcr.dvg.view.dsprojection.crystalreports.CrystalReportsViewerFrame;
import com.dcr.dvg.view.dsprojection.record.RecordResultsViewerFrame;
import com.dcr.dvg.view.dsprojection.sql.SqlResultsViewerFrame;
import com.dcr.dvg.view.dsprojection.meta.MetaViewerFrame;
import com.dcr.dvg.view.portal.PortalViewerFrame;

import com.dcr.dvg.view.portal.action.CopyPortalContentsAction;
import com.dcr.dvg.view.portal.action.ExitPortalViewerAction;
import com.dcr.dvg.view.portal.action.PortalViewerHelpAction;
import com.dcr.dvg.view.portal.action.NewChartViewerAction;
import com.dcr.dvg.view.portal.action.NewCrystalReportsViewerAction;
import com.dcr.dvg.view.portal.action.NewHiGridViewerAction;
import com.dcr.dvg.view.portal.action.NewMetaViewerAction;
import com.dcr.dvg.view.portal.action.NewPlexusViewerAction;
import com.dcr.dvg.view.portal.action.NewRecordViewerAction;
import com.dcr.dvg.view.portal.action.NewSqlResultsViewerAction;
import com.dcr.dvg.view.portal.action.PortalViewerHelpAction;
import com.dcr.dvg.view.portal.action.RefreshPortalViewerAction;
import com.dcr.dvg.view.portal.action.SavePortalViewerAction;
import com.dcr.dvg.view.portal.action.SaveAsPortalViewerAction;
import com.dcr.dvg.view.portal.action.ToggleShowSqlViewerAction;

import com.dcr.dvg.view.desktop.VPDesktopViewer;
import com.dcr.dvg.view.dsprojection.DataSourceViewerFrame;
import com.dcr.dvg.view.dsprojection.chart.ChartResultsViewerFrame;
import com.dcr.dvg.view.dsprojection.plexus.PlexusResultsViewerFrame;
import com.dcr.dvg.util.throwable.DataSourceCouldNotBeOpenedException;
import com.dcr.dvg.util.throwable.DVError;
import com.dcr.dvg.util.throwable.DVException;
import com.dcr.dvg.util.throwable.FileBeingOpenedDoesNotExistException;
import com.dcr.dvg.util.throwable.NoDataSourceSelectedForQueryExecutionException;

public class PortalViewer extends VPDesktopViewer implements InternalFrameListener {

        protected PortalModel portalModel = null;

        protected PortalViewerFrame viewerController = null;

        protected ChartResultsViewerFrame chartViewerFrame = null;
        protected VPHDesktopHiGridViewerFrame tableViewerFrame = null;

/**
 * Constructor based on the viewer frame.
 * <p>
 * @param viewerController - PortalViewerFrame
 */
public PortalViewer(PortalViewerFrame viewerController) {

        super();

        this.viewerController = viewerController;
}


/**
 * Adds a new external viewer.
 * <p>
 * @param viewerModel - ViewerModel
 * @param frame - DataSourceViewerFrame
 * @param dataSource - MDbNodeTreeData
 */
```

```
public void addNewExternalViewer(ViewerModel viewerModel, DataSourceViewerFrame frame,
MDbNodeTreeData dataSource) {

        showExternalViewer(viewerModel, frame, dataSource);
}

/**
 * Adds a new internal viewer.
 * <p>
 * @param viewerModel - DataSourceViewerModel
 * @param viewerFrame - DataSourceViewerFrame
 * @param dataSource - MDbNodeTreeData
 */
public void addNewInternalViewer(DataSourceViewerModel viewerModel,
DataSourceViewerFrame viewerFrame, MDbNodeTreeData dataSource)
  throws DVException {

        MDbNodeTreeData targetTreeDataSource = dataSource;
        if (targetTreeDataSource == null) {
                MDbNodeTreeData viewerModelDataSource =
((DataSourceViewerModel)viewerModel).getDataModel();
                if (viewerModelDataSource != null && viewerModelDataSource.isReady())
                        targetTreeDataSource = viewerModelDataSource;
                else
                        targetTreeDataSource =
(MDbNodeTreeData)getSelectedDataSource().clone();
        }

        if (targetTreeDataSource != null) {
                getPortalModel().addViewer(((DataSourceViewerModel)viewerModel));

        viewerModel.setDataModelNoNotify((MDbNodeTreeData)targetTreeDataSource);

                showDbViewer(((DataSourceViewerModel)viewerModel), viewerFrame,
targetTreeDataSource);
        } else
                fireDbResultsViewerException(new
NoDataSourceSelectedForQueryExecutionException());
}

/**
 * Adds a new viewer.
 * <p>
 * @param viewerModel - DataSourceViewerModel
 */
public void addNewViewer(ViewerModel viewerModel) {

        DataSourceViewerFrame frame = null;
        PortalViewerFrame viewerController = (PortalViewerFrame)getViewerController();
        JLayeredPane desktopView = getDesktopView();

        if (viewerModel instanceof MViewerHiGrid)
                frame = new VPHDesktopHiGridViewerFrame(viewerController, desktopView);
        else if (viewerModel instanceof MViewerChart)
                frame = new ChartResultsViewerFrame(viewerController, desktopView);
        else if (viewerModel instanceof MViewerPlexus)
                frame = new PlexusResultsViewerFrame(viewerController, desktopView);
        else if (viewerModel instanceof MViewerRecord)
                frame = new RecordResultsViewerFrame(viewerController, desktopView);
        else if (viewerModel instanceof MViewerCrystalReports)
                frame = new CrystalReportsViewerFrame(viewerController,
getDesktopView());
        else if (viewerModel instanceof MViewerSql)
                frame = new SqlResultsViewerFrame(viewerController, getDesktopView());
        else if (viewerModel instanceof MViewerMeta)
                frame = new MetaViewerFrame(viewerController, getDesktopView());

        addNewViewerOperation(viewerModel, frame, null);
}

/**
 * Adds a new viewer (basic operation).
 * <p>
 * @param viewerModel - DataSourceViewerModel
 * @param viewerFrame - DataSourceViewerFrame
 * @param dataSource - MDbNodeTreeData
```

```
 */
public void addNewViewerBasic(ViewerModel viewerModel, DataSourceViewerFrame
viewerFrame, MDbNodeTreeData dataSource)
  throws DVException {

        if (viewerModel instanceof DataSourceViewerModel)
                addNewInternalViewer((DataSourceViewerModel)viewerModel, viewerFrame,
dataSource);
        else
                addNewExternalViewer(viewerModel, viewerFrame, dataSource);
}

/**
 * Adds a new viewer.
 * <p>
 * @param viewerModel - DataSourceViewerModel
 * @param viewerFrame - DataSourceViewerFrame
 * @param dataSource - MDbNodeTreeData
 */
public void addNewViewerOperation(ViewerModel viewerModel, DataSourceViewerFrame
frame, MDbNodeTreeData dataSource) {

        String operationName = "Add New " + viewerModel.getModelTypeName() + " Viewer";

        setCursor(Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));
        fireOperationBegun(operationName);

        try {
                addNewViewerBasic(viewerModel, frame, dataSource);
        } catch (Exception exception) {
                if (exception instanceof NullPointerException)
                        fireOperationFailed(operationName + " DUE TO Data source setup
improperly.");
                else
                        fireOperationFailed(operationName + " DUE TO " +
exception.getMessage());
        } catch (Error error) {
                fireOperationFailed(operationName + " DUE TO " + error.getMessage());
        }

        fireOperationFinished(operationName);
        setCursor(Cursor.getPredefinedCursor(Cursor.DEFAULT_CURSOR));
}

/**
 * Add a sub frame for the specified viewer type.
 * <p>
 * @param viewerModel - ViewerModel
 * @param frame - DataSourceViewerFrame - the frame to add
 */
public void addSubViewerFrame(ViewerModel viewerModel, DataSourceViewerFrame frame) {

        if (viewerModel instanceof DataSourceViewerModel)
                frame.setViewerModel((DataSourceViewerModel)viewerModel);

        frame.addPropertyChangeListener(this);
        addPropertyChangeListener(frame);
        viewerModel.getPortal().addPropertyChangeListener((PortalViewerFrame)frame.getP
ortalViewerFrame());

        if (viewerModel instanceof DataSourceViewerModel)

        ((DataSourceViewerModel)viewerModel).addDataModelListener((MDbDataModelListener
)((DataSourceViewerFrame)frame).getViewer());

        viewerModel.addPropertyChangeListener(((DataSourceViewerFrame)frame).getViewer(
));

        frame.setClosable(true);

        applyDeskTopContextTo(frame);
}

/**
 * Apply the desktop context to the specified frame.
 * <p>
```

```
 * @param frame - DVGInternalFrame
 */
public void applyDeskTopContextTo(DVGInternalFrame frame) {

        ((DVGDesktop)getDesktopView()).applyDeskTopContextTo(frame);
}


/**
 * Close the viewer.
 */
public void close() {

        SessionModel.getSession().getDVGSharedInstance().closeSessionWindow();
}


/**
 * Copy the contents of a saved portal.
 */
public void copyContentsAction() {

        notifyOperationBegun("Copy contents of a saved portal");

        try {
                PortalModel sourcePortalModel = loadPortalFile();

                if (sourcePortalModel != null) {
                        Vector newViewers =
getPortalModel().copyContentsOf(sourcePortalModel);
                        if (! newViewers.isEmpty()) {
                                // NOT YET IMPLEMENTED
//                              Enumeration sourceViewersList = newViewers.elements();
//                              while (sourceViewersList.hasMoreElements())
//
        viewViewer((MViewer)sourceViewersList.nextElement());
                        }

                        notifyOperationFinished("Copy contents of a saved portal");
                }
        } catch (Exception event) {
                notifyOperationFailed("Copy contents of a saved portal");
                System.err.println("PortalModel Error in copying contents of Portal:");
                System.err.println("     " + event.getClass() + ":  " +
event.getMessage());
//              throw event;
        }
}

/**
 * Exit the portal.
 */
public void exitAction() {

        getPortalViewerFrame().dispose();
}


/**
 * Fetch the list of actions supported by this
 * editor.  It is implemented to return the list
 * of actions supported by the embedded JTextComponent
 * augmented with the actions defined locally.
 */
public Action[] getActions() {

        Action[] defaultActions = {
            new CopyPortalContentsAction(this),
            new SavePortalViewerAction(this),
            new SaveAsPortalViewerAction(this),
            new RefreshPortalViewerAction(this),
            new ExitPortalViewerAction(this),

            new NewChartViewerAction(this),
            new NewCrystalReportsViewerAction(this),
            new NewHiGridViewerAction(this),
```

```
            new NewMetaViewerAction(this),
            new NewPlexusViewerAction(this),
            new NewRecordViewerAction(this),
            new NewSqlResultsViewerAction(this),
            new ToggleShowSqlViewerAction(this),
            new PortalViewerHelpAction(this)
        };

        Action[] moreActions = TextAction.augmentList(super.getActions(),
defaultActions);
        return TextAction.augmentList(((DVGDesktop)getDesktopView()).getActions(),
moreActions);
}


/**
 * Gets the controller for this resource-employing component.
 * <p>
 * @return viewer - DVGDesktopViewer
 */
public DVGDesktopViewer getCommandDesktopViewer() {

        return ((PortalViewerFrame)getViewerController())
                                        .getCommandDesktopViewer();
}


/**
 * Gets the Data Source Directory.
 * <p>
 * @param dataSourceDirectory - DataSourceDirectory
 */
public DataSourceDirectory getDataSourceDirectory() {

        return ((DataSourceDirectoryControllerFrame)

        ((DVGDesktopViewer)getCommandDesktopViewer()).getDataSourceDirectoryControllerF
rame())
                                        .getTypedViewer().getDataSourceDirectory();
}

/**
 * Gets the desk top view.
 * <p>
 * @return desktopView - JLayeredPane
 */
public JLayeredPane getDesktopView() {

        return desktopView;
}


/**
 * Gets the portal's model.
 * <p>
 * @return portalModel - PortalModel
 */
public PortalModel getPortalModel() {

        return portalModel;
}


/**
 * Gets the frame for this viewer.
 * <p>
 * @return frame - PortalViewerFrame
 */
public PortalViewerFrame getPortalViewerFrame() {

        return ((PortalViewerFrame)getViewerController());
}


/**
 * Gets the selected Data Source from the Data Source Directory.
```

```
 * <p>
 * @param selectedDataSource - MDbNodeTreeData
 */
public MDbNodeTreeData getSelectedDataSource()
  throws DVException {

        return (MDbNodeTreeData)getDataSourceDirectory().getSelectedDataSource();
}


/**
 * Gets the Transcript Controller frame.
 * <p>
 * @return frame - TranscriptControllerFrame
 */
public TranscriptControllerFrame getTranscriptControllerFrame() {

        return
((DVGDesktopViewer)getCommandDesktopViewer()).getTranscriptControllerFrame();
}


/**
 * Gets the controller for this viewer.
 * <p>
 * @return frame - PortalViewerFrame
 */
public PortalViewerFrame getViewerController() {

        return viewerController;
}


/**
 * Initialized this viewer.
 */
public boolean initialize() {

        if (portalModel == null)
                portalModel = new PortalModel();

        // desktopView must be set before initializing this class with the view
        desktopView = new DVGDesktop(this);

        initialize("PortalViewer", (DVGDesktop)desktopView);

        commandsSetEnabled(true);

        return true;
}


/**
 * Implements InternalFrameListener.
 * <p>
 * Do nothing here.
 */
public void internalFrameActivated(InternalFrameEvent e) {
}


/**
 * Implements InternalFrameListener.
 * <p>
 * Do nothing here.
 */
public void internalFrameClosed(InternalFrameEvent event) {
}


/**
 * Implements InternalFrameListener.
 * <p>
 * Do nothing here.
 */
public void internalFrameClosing(InternalFrameEvent e) {
```

```
}


/**
 * Implements InternalFrameListener.
 * <p>
 * Do nothing here.
 */
public void internalFrameDeactivated(InternalFrameEvent e) {
}


/**
 * Implements InternalFrameListener.
 * <p>
 * Do nothing here.
 */
public void internalFrameDeiconified(InternalFrameEvent e) {
}


/**
 * Implements InternalFrameListener.
 * <p>
 * Do nothing here.
 */
public void internalFrameIconified(InternalFrameEvent e) {
}


/**
 * Implements InternalFrameListener.
 * <p>
 * Do nothing here.
 */
public void internalFrameOpened(InternalFrameEvent e) {
}


/**
 * Load a portal (model) from a file.
 * <p>
 * @return sourcePortal - PortalModel - the model of the loaded Portal
 */
public PortalModel loadPortalFile() {

        PortalModel sourcePortal = null;

        File file = getFileToOpen();
        if (file != null) {
                if (file.exists()) {
                        try {
                                FileInputStream fin = new FileInputStream(file);
                                ObjectInputStream stream = new ObjectInputStream(fin);
                                sourcePortal = (PortalModel)stream.readObject();
                        } catch (IOException io) {
                                // should put in status panel
                                System.err.println("IOException: " + io.getMessage());
                        } catch (ClassNotFoundException cnf) {
                                // should put in status panel
                                System.err.println("Class not found: " +
cnf.getMessage());
                        }

                        revalidate();
                } else
                        fireExceptionRaised(new
FileBeingOpenedDoesNotExistException(file.getName()));
        }

        return sourcePortal;
}

/**
 * Launch a new chart viewer frame.
 */
```

```
public void newChartViewer() {

        newChartViewer(null);
}

/**
 * Launch a new chart viewer frame.
 * <p>
 * @param dataSource - MDbNodeTreeData - data source to apply to the viewer
 */
public void newChartViewer(MDbNodeTreeData dataSource) {

        addNewViewerOperation(new MViewerChart(getPortalModel()), new
ChartResultsViewerFrame((PortalViewerFrame)getViewerController(), getDesktopView()),
dataSource);
}

/**
 * Launch a new plexus viewer frame.
 */
public void newCrystalReportsViewer() {

        newCrystalReportsViewer(null);
}


/**
 * Launch a new Crystal Reports viewer frame.
 * <p>
 * @param dataSource - MDbNodeTreeData - data source to apply to the viewer
 */
public void newCrystalReportsViewer(MDbNodeTreeData dataSource) {

        addNewViewerOperation(new MViewerCrystalReports(getPortalModel()), new
CrystalReportsViewerFrame((PortalViewerFrame)getViewerController(), getDesktopView()),
dataSource);
}


/**
 * Launch a new meta viewer frame.
 */
public void newMetaViewer() {

        newMetaViewer(null);
}

/**
 * Launch a new meta viewer frame.
 * <p>
 * @param dataSource - MDbNodeTreeData - data source to apply to the viewer
 */
public void newMetaViewer(MDbNodeTreeData dataSource) {

        addNewViewerOperation(new MViewerMeta(getPortalModel()), new
MetaViewerFrame((PortalViewerFrame)getViewerController(), getDesktopView()),
dataSource);
}

/**
 * Launch a new plexus viewer frame.
 */
public void newPlexusViewer() {

        newPlexusViewer(null);
}


/**
 * Launch a new plexus viewer frame.
 * <p>
 * @param dataSource - MDbNodeTreeData - data source to apply to the viewer
 */
public void newPlexusViewer(MDbNodeTreeData dataSource) {
```

```
        addNewViewerOperation(new MViewerPlexus(getPortalModel()), new
PlexusResultsViewerFrame((PortalViewerFrame)getViewerController(), getDesktopView()),
dataSource);
}


/**
 * Launch a new chart viewer frame.
 */
public void newRecordViewer() {

        newRecordViewer(null);
}

/**
 * Launch a new chart viewer frame.
 * <p>
 * @param dataSource - MDbNodeTreeData - data source to apply to the viewer
 */
public void newRecordViewer(MDbNodeTreeData dataSource) {

        addNewViewerOperation(new MViewerRecord(getPortalModel()), new
RecordResultsViewerFrame((PortalViewerFrame)getViewerController(), getDesktopView()),
dataSource);
}

/**
 * Launch a new SQL viewer frame.
 */
public void newSqlViewer() {

        newSqlViewer(null);
}

/**
 * Launch a new SQL viewer frame.
 * <p>
 * @param dataSource - MDbNodeTreeData - data source to apply to the viewer
 */
public void newSqlViewer(MDbNodeTreeData dataSource) {

        addNewViewerOperation(new MViewerSql(getPortalModel()), new
SqlResultsViewerFrame((PortalViewerFrame)getViewerController(), getDesktopView()),
dataSource);
}

/**
 * Launch a new grid viewer frame.
 */
public void newTableViewer() {

        newTableViewer(null);
}

/**
 * Launch a new grid viewer frame.
 * <p>
 * @param dataSource - MDbNodeTreeData - data source to apply to the viewer
 */
public void newTableViewer(MDbNodeTreeData dataSource) {

        addNewViewerOperation(new MViewerHiGrid(getPortalModel()), new
VPHDesktopHiGridViewerFrame((PortalViewerFrame)getViewerController(),
getDesktopView()), dataSource);
}

/**
 * Notify that an operation has begun.
 * <p>
 * @param operationName - String - name of the operation that is beginning
 */
public void notifyOperationBegun(String operationName) {

        getTranscriptControllerFrame().fireOperationBegun(operationName);
}
```

```
/**
 * Notify that an operation has canceled.
 * <p>
 * @param operationName - String - name of the operation that is canceled
 */
public void notifyOperationCanceled(String operationName) {

	getTranscriptControllerFrame().fireOperationCanceled(operationName);
}


/**
 * Notify that an operation has failed.
 * <p>
 * @param operationName - String - name of the operation that has failed
 */
public void notifyOperationFailed(String operationName) {

	getTranscriptControllerFrame().fireOperationFailed(operationName);
}


/**
 * Notify that an operation has finished.
 * <p>
 * @param operationName - String - name of the operation that has finished
 */
public void notifyOperationFinished(String operationName) {

	getTranscriptControllerFrame().fireOperationFinished(operationName);
}


/**
 * Called when a bound property is changed,
 * <p>
 * @param evt PropertyChangeEvent
 */
public void propertyChange(PropertyChangeEvent event) {

	tracePropertyChange(event);

	if (event.getPropertyName().equals("Connection Available")) { // e.g., SQL
command executed
		Object value = event.getNewValue();
		if (value instanceof MDbNodeTreeData)
			viewDataBaseConnection((MDbNodeTreeData)value);
		else if (value instanceof MDbTreeProperties)
			viewDataBaseConnection((MDbTreeProperties)value);
	} else if (event.getPropertyName().equals("DB Results Viewer Frame Exception"))
		fireDbResultsViewerException((DVException)event.getNewValue());
	else if (event.getPropertyName().equals("DB Results Viewer Frame Error"))
		fireDbResultsViewerError((DVError)event.getNewValue());
	else if (event.getPropertyName().equals("isClosed"))
		finalizeListenerIsClosed(event.getSource());
	else if (event.getPropertyName().equals(PortalModel.VIEWER_MODEL_ADDED))

	//viewDataBaseConnection(((MViewer)event.getNewValue()).getDataModel());
		event = event; //
}


/**
 * Refresh the portal.
 */
public void refreshPortal() {

	try {
		getPortalModel().refreshPortal();

		Vector sourceViewers = getPortalModel().getViewers();

		if (sourceViewers.isEmpty())
			return;
```

```
                Enumeration sourceViewersList = sourceViewers.elements();
                while (sourceViewersList.hasMoreElements()) {
                        DataSourceViewerModel viewerModel =
(DataSourceViewerModel)sourceViewersList.nextElement();

                        //viewerModel.getTreeProperties().notifyObservers();
                        MDbTreeProperties dataModel =
viewerModel.getDataModelComponent();
                        dataModel.fireJCDataEvent(new
MDbJCDataEvent((Object)dataModel));
                }
        } catch (Exception event) {
                System.err.println(getClass() + "Error in refreshing the portal:");
                System.err.println("     " + event.getClass() + ":  " +
event.getMessage());
        }
}


/**
 * Saves the portal.
 */
public void saveAction() {

        try {
                getPortalModel().savePortal();
        } catch (Exception event) {
                System.err.println("PortalModel Error in saving Portal:");
                System.err.println("     " + event.getClass() + ":  " +
event.getMessage());
        }
}


/**
 * Save As the portal.
 */
public void saveAsAction() {

        File file = getFileToSave();

        if (file != null)
                try {
                        getPortalModel().saveAsPortal(getFileToSave());
                } catch (Exception event) {
                        System.err.println("PortalModel Error in saving Portal:");
                        System.err.println("     " + event.getClass() + ":  " +
event.getMessage());
                }
}


/**
 * Sets the portal's model.
 */
public void setPortalModel(PortalModel portalModel) {

        this.portalModel = portalModel;
}


/**
 * Show a new db viewer.
 * <p>
 * @param viewerModel - DataSourceViewerModel
 * @param frame - DataSourceViewerFrame
 * @param treeDataSource - MDbNodeTreeData
 */
public void showDbViewer(DataSourceViewerModel viewerModel, DataSourceViewerFrame
frame, MDbNodeTreeData treeDataSource) {

        addSubViewerFrame(viewerModel, frame);
        frame.addInternalFrameListener(getPortalModel());

        InternalResultsViewer viewer = ((InternalResultsViewer)frame.getViewer());
        viewer.setDataSourceCustomizer();
```

```
        if (treeDataSource != null) {
                viewer.setDataSource(treeDataSource);

                viewer.setViewControllerTitle(treeDataSource.getModelName());
        } else {
                fireDbResultsViewerException((DVException)new
DataSourceCouldNotBeOpenedException());
                viewer.setViewControllerTitle(
                  MessageKit.getMessageText("DBResultsViewer",
"DataSourceCouldNotBeOpened")
                );
        }

        // check to see if data source was NOT set
        if (viewer.getTreeDataSourceData() == null)
                fireDbResultsViewerException((DVException)new
DataSourceCouldNotBeOpenedException());

        frame.setVisible(true);

        frame.fireNewDbResultsViewerFrame();
}

/**
 * Show a new external viewer.
 * <p>
 * @param viewerModel - DataSourceViewerModel
 * @param frame - DataSourceViewerFrame
 * @param treeDataSource - MDbNodeTreeData
 */
public void showExternalViewer(ViewerModel viewerModel, DataSourceViewerFrame frame,
MDbNodeTreeData dataSource) {

        addSubViewerFrame(viewerModel, frame);
        frame.addInternalFrameListener(getPortalModel());

        frame.setVisible(true);

        frame.fireNewDbResultsViewerFrame();
}

/**
 * Notifies that a new data base connection was established.
 */
public void viewDataBaseConnection() {

        traceMessage(getClass() + "::viewDataBaseConnection");

        firePropertyChange("SQL Query Available", null, VCPlexus.getTestEdges());

        refresh();
}


/**
 * View the data source using a table, that is grid, view.
 * <p>
 * @param dataSource - MDbNodeTreeData
 */
public void viewDataBaseConnection(MDbNodeTreeData dataSource) {

        traceMessage(getClass() + "  >>  viewDataBaseConnection for " + dataSource);

        addNewViewerOperation(new MViewerHiGrid(getPortalModel(), dataSource), new
VPHDesktopHiGridViewerFrame((PortalViewerFrame)getViewerController(),
getDesktopView()), null);

        refresh();
}


/**
 * Update the plexus view.
 * <p>
 * @param treeProperties - MDbTreeProperties
```

```
 */
public void viewDataBaseConnection(MDbTreeProperties treeProperties) {

        traceMessage(getClass() + "  >>  viewDataBaseConnection");

        firePropertyChange("SQL Query Available", null, (Object)treeProperties);

        ((DVGDesktop)getDesktopView()).cascadeFrames();

        refresh();
}


/**
 * View a viewer.
 * <p>
 * @param viewerModel - ViewerModel - model of the viewer
 */
public void viewViewer(ViewerModel viewerModel) {

        PortalViewerFrame hostFrame = (PortalViewerFrame)getViewerController();
        DataSourceViewerFrame frame = null;

        if (viewerModel instanceof MViewerHiGrid)
                frame = new VPHDesktopHiGridViewerFrame(hostFrame, getDesktopView());
        else if (viewerModel instanceof MViewerChart)
                frame = new ChartResultsViewerFrame(hostFrame, getDesktopView());
        else if (viewerModel instanceof MViewerPlexus)
                frame = new PlexusResultsViewerFrame(hostFrame, getDesktopView());
        else if (viewerModel instanceof MViewerRecord)
                frame = new RecordResultsViewerFrame(hostFrame, getDesktopView());
        else if (viewerModel instanceof MViewerCrystalReports)
                frame = new CrystalReportsViewerFrame(hostFrame, getDesktopView());
        else if (viewerModel instanceof MViewerSql)
                frame = new SqlResultsViewerFrame(hostFrame, getDesktopView());
        else if (viewerModel instanceof MViewerMeta)
                frame = new MetaViewerFrame(hostFrame, getDesktopView());

        if (viewerModel instanceof DataSourceViewerModel)
                showDbViewer((DataSourceViewerModel)viewerModel, frame,
((DataSourceViewerModel)viewerModel).getDataModel());
        else
                showExternalViewer(viewerModel, frame, null);
}
}

package com.dcr.dvg.view.portal;
/**
 * @(#)PortalViewerFrame.java
 * <p>
 * *****************************************************************************
 * *****************************************************************************
 * <p>
 * The <code>PortalViewerFrame</code> is the frame for a <code>PortalViewer</code>.
 * <p>
 * @author            Edward L. Stull
 *      @version 1.13
 * @since             JDK 2
 */
//34567890123456789012345678901234567890123456789*1234567890123456789012345678901234567890

import java.awt.Dimension;
import java.beans.PropertyChangeEvent;
import java.beans.PropertyVetoException;
import java.util.Enumeration;
import java.util.Vector;

import javax.swing.JLayeredPane;
import javax.swing.JInternalFrame;

import com.dcr.dve.model.mdb.MDbNodeProperties;
import com.dcr.dve.model.mdb.MDbNodeTreeData;
import com.dcr.dve.model.mcommand.mccompiler.mccparser.Parse;
import com.dcr.dve.view.vprocess.vphigrid.VPHDesktopHiGridViewerFrame;

import com.dcr.dvg.model.portal.directory.PortalModel;
```

```
import com.dcr.dvg.model.viewer.ViewerModel;
import com.dcr.dvg.model.viewer.DataSourceViewerModel;
import com.dcr.dvg.model.viewer.MViewerChart;
import com.dcr.dvg.model.viewer.MViewerCrystalReports;
import com.dcr.dvg.model.viewer.MViewerHiGrid;
import com.dcr.dvg.model.viewer.MViewerRecord;
import com.dcr.dvg.model.viewer.MViewerPlexus;
import com.dcr.dvg.util.throwable.DVError;
import com.dcr.dvg.util.throwable.DVException;
import com.dcr.dvg.util.throwable.NoDataSourceSelectedForQueryExecutionException;

import com.dcr.dvg.view.component.desktop.DVGDesktop;
import com.dcr.dvg.view.component.desktop.DVGInternalFrame;
import com.dcr.dvg.view.controller.datasource.DataSourceDirectory;
import com.dcr.dvg.view.controller.datasource.DataSourceDirectoryControllerFrame;
import com.dcr.dvg.view.controller.portal.PortalDirectoryController;
import com.dcr.dvg.view.controller.portal.PortalDirectoryControllerFrame;
import com.dcr.dvg.view.desktop.DVGDesktopViewer;
import com.dcr.dvg.view.dsprojection.DataSourceViewerFrame;
import com.dcr.dvg.view.dsprojection.chart.ChartResultsViewerFrame;
import com.dcr.dvg.view.dsprojection.crystalreports.CrystalReportsViewerFrame;
import com.dcr.dvg.view.dsprojection.plexus.PlexusResultsViewerFrame;
import com.dcr.dvg.view.dsprojection.record.RecordResultsViewerFrame;

public class PortalViewerFrame extends DVGInternalFrame {

        protected boolean hasExceptionOrErrorOccured = false;

        protected MDbNodeTreeData dbConnection = null;

/**
 * Constructor.
 * <p>
 * @param desktopView - JLayeredPane - the parent container
 * @param frameDimension - Dimension - the dimension of the frame
 */
public PortalViewerFrame(JLayeredPane desktopView, Dimension frameDimension) {

        super(desktopView, frameDimension);
}

/**
 * Find the frame with the specified name (title).
 * <p>
 * @param frameNameObject  Object
 * @return frane  DataSourceViewerFrame
 */
public DataSourceViewerFrame findFrameWithName(Object frameNameObject) {

        JInternalFrame[] frames = getAllFrames();

        for (int i = 0; i < frames.length; i++)
                if (frames[i] instanceof DataSourceViewerFrame
                                &&
frames[i].getTitle().equals((String)frameNameObject.toString()))
                        return (DataSourceViewerFrame)frames[i];

        throw new Error("INTERNAL ERROR: findFrameWithName could not find a DB Results
frame");
}

/**
 * Fire "DB Results Error" property change for an error.
 * <p>
 * @param error - DVError
 */
public void fireDbResultsError(DVError error) {

        traceMessage(getClass() + ">>fireDbResultsError =" + error);

        setExceptionOrErrorOccured();

        firePropertyChange("DB Results Error", null, error);
}
```

```
/**
 * Fire "DB Results Exception" property change for an exception.
 * <p>
 * @param exception - DVException
 */
public void fireDbResultsException(DVException exception) {

        if (TRACE)
                traceMessage(getClass() + "  >>  fireDbResultsException =" +
exception);

        setExceptionOrErrorOccured();

        firePropertyChange("DB Results Exception", null, exception);
}


/**
 * Fire that SQL query (frame) are available.
 * <p>
 * @param dataSource - MDbNodeProperties
 */
public void fireDBSqlQueryAvailable(MDbNodeProperties dataSource) {

        firePropertyChange("SQL Query Available", null, (Object)dataSource);
}


/**
 * Fire that SQL query results (frame) are available.
 * <p>
 * @param dataSource - MDbNodeProperties
 */
public void fireSqlQueryResultsAvailable() {

        firePropertyChange("SQL Query Results Available", null, new Boolean(true));
}


/**
 * Gets all JInternalFrames currently displayed in the
 * desktop. Returns iconified frames as well as expanded frames.
 * <p>
 * @return framesArray - an array of JInternalFrame objects
 */
public JInternalFrame[] getAllFrames() {

    int i, count;
    JInternalFrame[] results;
    Vector vResults = new Vector(10);
    Object next, tmp;

    count = getComponentCount();
    for (i = 0; i < count; i++) {
        next = getComponent(i);
        if (next instanceof JInternalFrame)
            vResults.addElement(next);
        else
            if (next instanceof JInternalFrame.JDesktopIcon) {
                tmp = ((JInternalFrame.JDesktopIcon) next).getInternalFrame();
                if (tmp != null)
                    vResults.addElement(tmp);
            }
    }

    results = new JInternalFrame[vResults.size()];
    vResults.copyInto(results);

    return results;
}


/**
 * Gets all frames on selected layers.
 * <p>
 * @return framesArray - an array of JInternalFrame objects
```

```
 */
public JInternalFrame[] getAllFramesOnMarkedLayers() {

        Vector frames = new Vector(10);

        JInternalFrame[] framesArray = new JInternalFrame[frames.size()];
        frames.copyInto(framesArray);

        return framesArray;
}

/**
 * Gets the controller for this resource-employing component.
 * <p>
 * @return viewer - DVGDesktopViewer
 */
public DVGDesktopViewer getCommandDesktopViewer() {

        return
(DVGDesktopViewer)((DVGDesktop)getDesktopView()).getCommandDesktopViewer();
}


/**
 * Gets the Data Source Directory (view).
 * <p>
 * @return dataSourceDirectory  DataSourceDirectory
 */
public DataSourceDirectory getDataSourceDirectory() {

        return
(DataSourceDirectory)((DataSourceDirectoryControllerFrame)getDesktopViewer().getDataSo
urceDirectoryControllerFrame()).getTypedViewer().getDataSourceDirectory();
}

/**
 * Gets the desktop viewer.
 * <p>
 * @return desktopViewer - DVGDesktopViewer
 */
public DVGDesktopViewer getDesktopViewer() {

        return (DVGDesktopViewer)((DVGDesktop)getDesktopView()).getController();
}

 /**
 * Gets the Portal Directory Controller.
 * <p>
 * @param portalDirectoryController - PortalDirectoryController
 */
public PortalDirectoryController getPortalDirectoryController() {

        return
(PortalDirectoryController)((PortalDirectoryControllerFrame)getDesktopViewer().getPort
alDirectoryControllerFrame()).getViewer();
}

/**
 * Gets the viewer.
 * <p>
 * @param portalViewer  PortalViewer
 */
public PortalViewer getPortalViewer() {

        return (PortalViewer)getViewer();
}

/**
 * Gets the portal viewer.
 * <p>
 * @return portalViewer PortalViewer
 */
public PortalViewer getResultsViewer() {

        return (PortalViewer)getViewer();
}
```

```
 /**
 * Gets the selected portal from the portal directory viewer.
 * <p>
 * @return selectedPortal - PortalModel
 */
public PortalModel getSelectedPortal() {

        return (PortalModel)getPortalDirectoryController().getSelectedPortal();
}


/**
 * Answer if an exception or error has occured during results genteration.
 * <p>
 * @return hasExceptionOrErrorOccured - boolean
 */
public boolean hasExceptionOrErrorOccured() {

        return hasExceptionOrErrorOccured;
}

/**
 * Called when a bound property is changed,
 * <p>
 * @param evt - PropertyChangeEvent
 */
public void propertyChange(PropertyChangeEvent event) {

        tracePropertyChange(event);

        if (event.getPropertyName().equals("Sql Query Pending"))
                viewPortal();
        else if (event.getPropertyName().equals("DB Results Viewer Exception"))
                fireDbResultsException((DVException)event.getNewValue());
        else if (event.getPropertyName().equals("DB Results Viewer Error"))
                fireDbResultsError((DVError)event.getNewValue());
        else if (event.getPropertyName().equals(PortalModel.VIEWER_MODEL_ADDED))
                getViewer().propertyChange(event);
        else
                super.propertyChange(event);
}


/**
 * Sets an exception or error has occured during results genteration.
 */
public void setExceptionOrErrorOccured() {

        hasExceptionOrErrorOccured = true;
}

 /**
 * This is the dynamic creation of a viewer in that it is created based on the
 * semantics of a portal which are not known a priori.
 */
public void viewPortal() {

        viewPortal(new PortalModel());
}


/**
 * This is the dynamic creation of a viewer in that it is created based on the
 * semantics of a portal which are not known a priori.
 */
public void viewPortal(PortalModel portalModel) {

        setTitle(getSelectedPortal().getName() + " #" + getDbResultsFrameId());

        setViewer(new PortalViewer(this));
        getPortalViewer().addPropertyChangeListener(this);
        getPortalViewer().setPortalModel(portalModel);
        getPortalViewer().initialize();
        getPortalViewer().viewDataBaseConnection();
```

```
        setContentPane(getPortalViewer());

        int layer = DVGDesktop.OUTPUT_LAYER;
        setNewFrameBounds(layer);
        getDesktopView().add(this, new Integer(layer));

        if (! hasExceptionOrErrorOccured()) {
                // next line to clean problem with apparently JCTable ghost on desktop
                // hopefully will be fixed by vendors later
                getDesktopView().repaint();

                viewViewers(portalModel.getViewers());

                getResultsViewer().refresh();
                refresh();
                fireSqlQueryResultsAvailable();
        } else
                dispose();
}


/**
 * View a viewer.
 * <p>
 * @param viewerModel - ViewerModel
 */
public void viewViewer(ViewerModel viewerModel) {

        MDbNodeTreeData treeDataSource =
((DataSourceViewerModel)viewerModel).getDataModel();

        if (viewerModel instanceof MViewerHiGrid)
                getPortalViewer().showDbViewer((DataSourceViewerModel)viewerModel, new
VPHDesktopHiGridViewerFrame(this, getDesktopView()), treeDataSource);
        else if (viewerModel instanceof MViewerChart)
                getPortalViewer().showDbViewer((DataSourceViewerModel)viewerModel, new
ChartResultsViewerFrame(this, getDesktopView()), treeDataSource);
        else if (viewerModel instanceof MViewerRecord)
                getPortalViewer().showDbViewer((DataSourceViewerModel)viewerModel, new
RecordResultsViewerFrame(this, getDesktopView()), treeDataSource);
        else if (viewerModel instanceof MViewerPlexus)
                getPortalViewer().showDbViewer((DataSourceViewerModel)viewerModel, new
PlexusResultsViewerFrame(this, getDesktopView()), treeDataSource);
        else if (viewerModel instanceof MViewerCrystalReports)
                getPortalViewer().showExternalViewer(viewerModel, new
CrystalReportsViewerFrame(this, getDesktopView()), null);
}

/**
 * View the portal's viewers.
 * <p>
 * @param viewers - Vector
 */
public void viewViewers(Vector viewers) {

        if (viewers == null)
                return;

        Enumeration viewersList = viewers.elements();
        while (viewersList.hasMoreElements())
                viewViewer((DataSourceViewerModel)viewersList.nextElement());
}
}
```

workflow

```
package com.dcr.dvg.view.controller.datasource;
/**
 *      @(#)DataSourceDirectoryController.java
 * <p>
 * ********************************************************************************
 * <p>
 * The <code>DataSourceDirectoryController</code> class is a view of the
 *       of a Data Source Directory with features for management.
 * <p>
 * @author              Edward L. Stull
 * @version 1.30
 * @since               JDK 2
 */
//345678901234567890123456789012345678901234567890123456789*12345678901234567890123456789

import java.awt.Cursor;
import java.awt.Dimension;
import java.awt.FileDialog;
import java.awt.Frame;
import java.awt.Image;
import java.beans.Beans;
import java.beans.PropertyChangeEvent;
import java.io.File;
import java.io.IOException;
import java.util.Hashtable;

import javax.swing.Action;
import javax.swing.JLayeredPane;
import javax.swing.SwingConstants;
import javax.swing.text.TextAction;

import com.dcr.dve.model.mdb.MDbNodeTreeData;
import com.dcr.dve.model.muser.MUserContext;
import com.dcr.dve.view.vcomponent.VCBorderLayout;
import com.dcr.dve.view.vcomponent.vccontrol.VCControlBar;
import com.dcr.dve.view.vcomponent.VCSplitViewer;
import com.dcr.dve.view.vcomponent.vcdialog.VCOptionDialog;
import com.dcr.dve.view.vcomponent.vcpanel.VCScrollViewer;
import com.dcr.dve.view.vprocess.IVPDataSourceCustomizerController;
import com.dcr.dve.view.vprocess.IVPView;
import com.dcr.dve.view.vprocess.VPDataSourceCustomizer;
import com.dcr.dve.view.vprocess.Viewer;

import com.dcr.dvg.model.datasource.directory.DataSourceDirectoryDataSourceNodeModel;
import com.dcr.dvg.model.datasource.directory.DataSourceFolderModel;
import com.dcr.dvg.model.datasource.directory.DataSourceDirectoryFolderNodeModel;
import com.dcr.dvg.model.datasource.directory.DataSourceDirectoryNodeModel;
import com.dcr.dvg.model.datasource.directory.DataSourceDirectoryModel;
import com.dcr.dvg.model.tree.ITreeFolderNodeModel;
import com.dcr.dvg.model.tree.explorer.IExplorerFolderModel;
import com.dcr.dvg.model.tree.explorer.TreeDirectoryModel;
import com.dcr.dvg.util.throwable.DVException;
import com.dcr.dvg.util.throwable.NoDataSourceSelectedException;
import com.dcr.dvg.util.throwable.NoDataSourceFolderSelectedException;
import com.dcr.dvg.util.throwable.DVShouldNotBeReachableInternalError;
import com.dcr.dvg.util.throwable.UnableToNewDataSourceDirectoryError;
import com.dcr.dvg.util.throwable.UnableToOpenDataSourceDirectoryFileException;
import com.dcr.dvg.util.throwable.UnableToSaveDataSourceDirectoryError;
import com.dcr.dvg.util.throwable.UnableToSaveDataSourceDirectoryFileException;

import com.dcr.dvg.view.controller.datasource.action.AddDataSourceAction;
import com.dcr.dvg.view.controller.datasource.action.AddDataSourceFolderAction;
import
com.dcr.dvg.view.controller.datasource.action.DataSourceDirectoryControllerInPlaceActi
on;
import
com.dcr.dvg.view.controller.datasource.action.DataSourceDirectoryControllerFolderActio
n;
import com.dcr.dvg.view.controller.datasource.action.EditDataSourceFolderAction;
import com.dcr.dvg.view.controller.datasource.action.EditDataSourceAction;
import com.dcr.dvg.view.controller.datasource.action.NewDataSourceDirectoryAction;
import com.dcr.dvg.view.controller.datasource.action.OpenDataSourceDirectoryAction;
import com.dcr.dvg.view.controller.datasource.action.SaveDataSourceDirectoryAction;
import com.dcr.dvg.view.controller.datasource.action.SaveAsDataSourceDirectoryAction;
```

```
import
com.dcr.dvg.view.controller.datasource.action.CopyDataSourceDirectoryElementAction;
import
com.dcr.dvg.view.controller.datasource.action.CutDataSourceDirectoryElementAction;
import
com.dcr.dvg.view.controller.datasource.action.DeleteDataSourceDirectoryElementAction;
import
com.dcr.dvg.view.controller.datasource.action.PasteDataSourceDirectoryElementAction;

import
com.dcr.dvg.view.controller.datasource.action.ToggleShowAsDataSourceDbResultsAction;

import com.dcr.dvg.view.controller.datasource.action.DataSourceDirectoryHelpAction;
import com.dcr.dvg.view.controller.datasource.action.ExitDataSourceDirectoryAction;
import
com.dcr.dvg.view.controller.datasource.exception.UnableToAddDataSourceException;
import
com.dcr.dvg.view.controller.datasource.exception.UnableToAddDataSourceFolderException;
import
com.dcr.dvg.view.controller.datasource.exception.UnableToCopyDataSourceDirectoryElemen
tException;
import
com.dcr.dvg.view.controller.datasource.exception.UnableToCutDataSourceDirectoryElement
Exception;
import
com.dcr.dvg.view.controller.datasource.exception.UnableToDeleteDataSourceDirectoryElem
entException;
import
com.dcr.dvg.view.controller.datasource.exception.UnableToNewDataSourceDirectoryExcepti
on;
import
com.dcr.dvg.view.controller.datasource.exception.UnableToPasteDataSourceDirectoryEleme
ntException;
import
com.dcr.dvg.view.controller.datasource.exception.UnableToSaveDataSourceDirectoryExcept
ion;
import
com.dcr.dvg.view.controller.datasource.exception.UnableToSaveAsDataSourceDirectoryExce
ption;
import com.dcr.dvg.view.controller.directory.DirectoryController;
import com.dcr.dvg.view.controller.directory.ExplorerDirectory;

import com.dcr.dvg.util.throwable.NoDataSourceFolderSelectedException;
import com.dcr.dvg.view.controller.iteration.IterationControllerFrame;
import com.dcr.dvg.view.dsprojection.DataSourceViewerFrame;

public class DataSourceDirectoryController
  extends DirectoryController
  implements IVPDataSourceCustomizerController {

        protected DataSourceViewerFrame previewDataSourceFrame = null;

        protected boolean showAsDbResults = false;
        protected boolean previewingOfDataSources = false;

        protected ExistingDataSourceFolderWizard existingDataSourceFolderPanel;
        protected NewDataSourceFolderWizard newDataSourceFolderPanel;
        //protected VPCommandConnectionExistingConnectionWizard
existingConnectionPanel;
        //protected VPCommandConnectionNewConnectionWizard newConnectionPanel;

        static String[] DataSourceOptionNames = { "Set", "Cancel" };

        protected boolean dialogActive;

/**
 * Constructor.
 */
public DataSourceDirectoryController() {

        super();
}


/**
```

```
 * Constructor based on the desktop.
 * <p>
 * @param desktopView - JLayeredPane
 */
public DataSourceDirectoryController(JLayeredPane desktopView) {

        super();

        setDesktopView(desktopView);

        initialize();
}


/**
 * Add a Data Source.
 */
public void actionAddDataSource(DataSourceDirectoryControllerFolderAction action) {

        applyOperation(action);
}


/**
 * Add a new folder.
 */
public void actionAddDataSourceFolder(DataSourceDirectoryControllerFolderAction
action) {

        applyOperation(action);
}


/**
 * Copy a directory element.
 */
public void
actionCopyDataSourceDirectoryNode(DataSourceDirectoryControllerFolderAction action) {

        applyOperation(action);
}

/**
 * Copy a new directory element.
 */
public void actionCutDataSourceDirectoryNode(DataSourceDirectoryControllerFolderAction
action) {

        applyOperation(action);
}

/**
 * Delete the selected Data Source Directory element Action.
 */
public void
actionDeleteDataSourceDirectoryNode(DataSourceDirectoryControllerFolderAction action)
{

        applyOperation(action);
}

/**
 * Edit a Data Source.
 */
public void actionEditDataSource(DataSourceDirectoryControllerInPlaceAction action) {

        applyOperation(action);
}

/**
 * Edit a Data Source Folder.
 */
public void actionEditDataSourceFolder(DataSourceDirectoryControllerInPlaceAction
action) {

        applyOperation(action);
```

```
}

/**
 * Get new Data Source Directory.
 */
public void actionNewDataSourceDirectory(DataSourceDirectoryControllerInPlaceAction
action) {

        applyOperation(action);
}

/**
 * Open a "saved" data source directory.
 */
public void actionOpenDataSourceDirectory(DataSourceDirectoryControllerInPlaceAction
action) {

        applyOperation(action);
}

/**
 * Paste an element to a folder.
 */
public void
actionPasteDataSourceDirectoryNode(DataSourceDirectoryControllerFolderAction action) {

        applyOperation(action);
}

/**
 * Save the data source directory.
 */
public void actionSaveAsDataSourceDirectory(DataSourceDirectoryControllerInPlaceAction
action) {

        applyOperation(action);
}

/**
 * Save the data source directory.
 */
public void actionSaveDataSourceDirectory(DataSourceDirectoryControllerInPlaceAction
action) {

        applyOperation(action);
}

/**
 * Save the connections.
 */
public void actionToggleShowAsDbResults() {

        if (! getShowAsDbResults())
                setShowAsDbResults(true);
        else
                setShowAsDbResults(false);
}


/**
 * Add a new portal folder.
 * <p>
 * @param folderNode - ITreeFolderNodeModel - folder to add
 */
public boolean addFolder(ITreeFolderNodeModel folderNode) {

        boolean completed = true;

        int selection = VCOptionDialog.showOptionDialog(
                this,
        //parentComponent
                newDataSourceFolderPanel.update(),    //message - object to display
                "New Data Source Folder Information",//title - title string for the
dialog
                VCOptionDialog.DEFAULT_OPTION,                //optionType -
YES_NO_OPTION or YES_NO_CANCEL_OPTION
```

```
                VCOptionDialog.QUESTION_MESSAGE,      //messageType - ERROR_MESSAGE,
INFORMATION_MESSAGE, etc
                null,                                                    //icon -
icon to display in the dialog
                AddElementOptionNames,                          //options - array of
possible choices
                AddElementOptionNames[1]);                      //initialValue - object
that is the default selection

        if(selection == VCOptionDialog.OK_OPTION) {
                setVisible(true);
                newDataSourceFolderPanel.getEditor().setFolder();


        newDataSourceFolderPanel.getEditor().getFolder().setOwner(MUserContext.getUserI
d());

                getDataSourceDirectoryModel().insertNode(
                  new
DataSourceDirectoryFolderNodeModel(newDataSourceFolderPanel.getEditor().getFolder()),
                  getFolderNodeOfSelectedDataSource()
                );

                getDataSourceDirectory().invalidate();

                updateView();
        } else
                completed = false;

        return completed;
}

/**
 * Add a new data source, that is, terminal to the tree.
 * <p>
 * @param folderNode - ITreeFolderNodeModel - folder to add to
 */
public boolean addTerminal(ITreeFolderNodeModel folderNode) {

        boolean completed = true;

        String title = getMessageUsingResource("DataSourceWizardDialogTitle");
        VPDataSourceCustomizer customizer = showDataSourceWizardDialog(true, title);

        if(customizer != null) {
                getDataSourceDirectoryModel()
                  .insertNode(
                    new
DataSourceDirectoryDataSourceNodeModel(customizer.getJCTreeData()),
                      folderNode
                );

                refresh();

                updateView();
        } else
                completed = false;

        return completed;
}

/**
 * Clear that a dialog is active.
 */
public void clearDialogActive() {

        dialogActive = false;
}


/**
 * Clear the results viewer.
 * <p>
 * DO NOTHING HERE
 */
public void clearView() {
```

```
}


/**
 * Copy a directory node.
 * <p>
 * @param folderNode - ITreeFolderNodeModel - folder to add to
 */
public boolean copyNode(ITreeFolderNodeModel folderNode) {

        boolean completed = true;

        try {

        getDataSourceDirectoryModel().copyToClipBoardNodeOf((DataSourceDirectoryNodeMod
el)getDataSourceDirectory().getSelectedNode());

                getDataSourceDirectory().invalidate();
        } catch (Exception exception) {
                fireDbResultsException(new
UnableToCopyDataSourceDirectoryElementException(exception));
                completed = false;
        }

        return completed;
}

/**
 * Cut a directory element.
 * <p>
 * @param folderNode - ITreeFolderNodeModel - folder to cut from
 */
public boolean cutNode(ITreeFolderNodeModel folderNode) {

        boolean completed = true;

        try {

        getDataSourceDirectoryModel().cutToClipBoardNodeOf((DataSourceDirectoryNodeMode
l)getDataSourceDirectory().getSelectedNode());

                getDataSourceDirectory().invalidate();

                //will reset clipboard ==> updateView();
        } catch (Exception exception) {
                fireDbResultsException(new
UnableToCutDataSourceDirectoryElementException(exception));
                completed = false;
        }

        return completed;
}

/**
 * Delete a directory element.
 * <p>
 * @param folderNode - ITreeFolderNodeModel - folder to delete from
 */
public boolean deleteNode(ITreeFolderNodeModel folderNode) {

        boolean completed = true;

        try {

        getDataSourceDirectoryModel().removeNode((DataSourceDirectoryNodeModel)getDataS
ourceDirectory().getSelectedNode());
                getDataSourceDirectory().refresh();

                updateView();
        } catch (Exception exception) {
                fireDbResultsException(new
UnableToDeleteDataSourceDirectoryElementException(exception));
                completed = false;
        }

        return completed;
```

```
}

/**
 * Edit a Data Source.
 * <p>
 * @return completed - boolean - success indicator
 */
public boolean editDataSource() {

        boolean completed = true;

        Object node = null;
        try {
                node = getDataSourceDirectory().getSelectedDataSourceNode();
        } catch (Exception exception) {
        }

        if (node instanceof DataSourceDirectoryDataSourceNodeModel) {
                try {
                        editDataSource((DataSourceDirectoryDataSourceNodeModel)node);

                        updateView();
                } catch (DVException exception) {
                        fireDbResultsException(new NoDataSourceSelectedException());
                        completed = false;
                }
        } else {
                fireDbResultsException(new NoDataSourceSelectedException());
                completed = false;
        }

        return completed;
}

/**
 * Edits a data source.
 * <p>
 * Bring up a dialog containing a new data source information panel.
 * <p>
 * @param dataSourceNode - DataSourceDirectoryDataSourceNodeModel - data source to
edit
 */
private void editDataSource(DataSourceDirectoryDataSourceNodeModel dataSourceNode)
  throws DVException {

        MDbNodeTreeData dataSource =
(MDbNodeTreeData)((DataSourceDirectoryDataSourceNodeModel)dataSourceNode).getTreeDataS
ource();

        notifyOperationBegun("Edit Data Source" + dataSource.getModelName());

        VPDataSourceCustomizer customizer
          = showDataSourceWizardDialog(
                false,
             "Edit Existing Data Source Information for "
               + "\"" + dataSource.getModelName() + "\"");

        if(customizer != null) {
                dataSourceNode.setTreeDataSource(customizer.getJCTreeData());
                getDataSourceDirectory().invalidate();

                notifyOperationFinished("Edit Data Source" +
dataSource.getModelName());
        } else
                notifyOperationCanceled("Edit Data Source" +
dataSource.getModelName());
}

/**
 * Edits a data source folder node.
 * <p>
 * Bring up a dialog containing a new connection information panel.
 * <p>
 * @param folderNode - DataSourceFolderModel - folder to edit
 */
private void editDataSourceFolder(DataSourceFolderModel folderNode) {
```

```
        notifyOperationBegun("Edit Data Source Folder" + folderNode.getName());

        int selection = VCOptionDialog.showOptionDialog(
                this,
        //parentComponent
                existingDataSourceFolderPanel.update(folderNode), //message - object to
display
                "Edit Connection Information for " + "\"" + folderNode.getName() +
"\"", //title - title string for the dialog
                VCOptionDialog.DEFAULT_OPTION,              //optionType -
YES_NO_OPTION or YES_NO_CANCEL_OPTION
                VCOptionDialog.QUESTION_MESSAGE,     //messageType - ERROR_MESSAGE,
INFORMATION_MESSAGE, etc
                null,                                                    //icon -
icon to display in the dialog
                EditElementOptionNames,                                  //options - array
of possible choices
                EditElementOptionNames[1]);                 //initialValue - object
that is the default selection

        if(selection == VCOptionDialog.OK_OPTION) {
                setVisible(true);
                existingDataSourceFolderPanel.getEditor().setFolder();

                notifyOperationFinished("Edit Data Source Folder" +
folderNode.getName());
        } else {
                notifyOperationCanceled("Edit Data Source Folder" +
folderNode.getName());
        }
}

/**
 * Edits a data source folder node.
 * <p>
 * Bring up a dialog containing a new data source information panel.
 */
public boolean editFolder() {

        boolean completed = true;

        fireClearDesktopStatus();

        Object node = getDataSourceDirectory().getSelectedDataSourceFolder();

        if (node instanceof DataSourceFolderModel) {
                editDataSourceFolder((DataSourceFolderModel)node);

                updateView();
        } else {
                fireDbResultsException(new NoDataSourceFolderSelectedException());
                completed = false;
        }

        return completed;
}

/**
 * Fetch the list of actions supported by this launcher.
 * @return the list of actions supported by the embedded JTextComponent
 *                              augmented with the actions defined locally.
 */
public Action[] getActions() {

        Action[] defaultActions = {
            new EditDataSourceFolderAction(this),
            new AddDataSourceFolderAction(this),

            new EditDataSourceAction(this),
            new AddDataSourceAction(this),

            new NewDataSourceDirectoryAction(this),
            new OpenDataSourceDirectoryAction(this),
            new SaveDataSourceDirectoryAction(this),
            new SaveAsDataSourceDirectoryAction(this),
```

```
            new CopyDataSourceDirectoryElementAction(this),
            new CutDataSourceDirectoryElementAction(this),
            new DeleteDataSourceDirectoryElementAction(this),
            new PasteDataSourceDirectoryElementAction(this),

            //new VPCommandConnectionTogglePreviewingOfConnectionsAction(this),
            new ToggleShowAsDataSourceDbResultsAction(this),

            new DataSourceDirectoryHelpAction(this),
            new ExitDataSourceDirectoryAction(this),
        };

        return TextAction.augmentList(super.getActions(), defaultActions);
    }

    /**
     * Gets the Data Source Customizer.
     */
    public VPDataSourceCustomizer getDataSourceCustomizer() {

        throw new DVShouldNotBeReachableInternalError(this);
    }


    /**
     * Gets the Data Source Directory view.
     * <p>
     * @return dataSourceDirectory - DataSourceDirectory
     */
    public DataSourceDirectory getDataSourceDirectory() {

        return ((DataSourceDirectoryView)getView()).getDataSourceDirectory();
    }


    /**
     * Gets the Data Source Directory.
     * <p>
     * @return directory - DataSourceDirectoryModel
     */
    public DataSourceDirectoryModel getDataSourceDirectoryModel() {

        return
    (DataSourceDirectoryModel)getDataSourceDirectory().getDataSourceDirectoryModel();
    }


    /**
     * Gets the Explorer Directory view.
     * <p>
     * @return explorerDirectory - ExplorerDirectory
     */
    public ExplorerDirectory getExplorerDirectory() {

        return getDataSourceDirectory();
    }


    /*
     * Gets the folder that owns this data source.
     * <p>
     * @return folder - DataSourceDirectoryFolderNodeModel
     */
    public DataSourceDirectoryFolderNodeModel getFolderNodeOfSelectedDataSource() {

        return
    (DataSourceDirectoryFolderNodeModel)getDataSourceDirectory().getFolderNodeOfSelectedDa
    taSource();
    }


    /**
     * Gets a message using a resource.
     * <p>
     * @return message = String
```

```
 */
public String getMessageUsingResource(String key) {

        return getMessageUsingResource("", getResourcesName(), getResources(), key,
"");
}


/**
 * Gets the type name of the node (e.g., Portal, Data Source) managed by the
controller.
 * <p>
 * @return nodeTypeName - String
 */
public String getNodeTypeName() {

        return "Data Source";
}

/**
 * Gets the exception for "no folder selected".
 * <p>
 * @return exception = DVException
 */
public DVException getNoFolderSelectedException() {

        return new NoDataSourceFolderSelectedException();
}


/**
 * Answer if dataSources are to be previewed.
 * <p>
 * @return previewingOfConnections - boolean
 */
public boolean getPreviewingOfDataSources() {

        return previewingOfDataSources;
}


/**
 * Gets the selected Data Source from the directory.
 * <p>
 * @param selectedDataSource = MDbNodeTreeData - return null if no Data Source is
selected
 */
public MDbNodeTreeData getSelectedDataSource()
  throws DVException {

        return getDataSourceDirectory().getSelectedDataSource();
}


/**
 * Gets the selected Data Source Folder from the directory.
 * <p>
 * @param selectedFolder = MDbNodeTreeData - return null if no Data Source Folder is
selected
 */
public DataSourceFolderModel getSelectedDataSourceFolder() {

        return getDataSourceDirectory().getSelectedDataSourceFolder();
}


/**
 * Answer if the Data Source is to be displayed.
 * <p>
 * @return showAsDbResults - boolean
 */
public boolean getShowAsDbResults() {

        return showAsDbResults;
}
```

```
/**
 * Gets the Tree Directory Model.
 * <p>
 * @return directoryModel - TreeDirectoryModel
 */
public TreeDirectoryModel getTreeDirectoryModel() {

	return getDataSourceDirectoryModel();
}


/**
 * Initializes.
 * <p>
 * @return success - boolean
 */
public boolean initialize() {

	initialize("DataSourceDirectoryController", new DataSourceDirectoryView());

	getDataSourceDirectory().addTreeSelectionListener(new
DataSourceDirectorySelectionListener(this));

	getDataSourceDirectory().expandRow(1);

	existingDataSourceFolderPanel = new ExistingDataSourceFolderWizard(this);
	newDataSourceFolderPanel = new NewDataSourceFolderWizard(this);

	return true;
}


/**
 * Answers true if a dialog is active.
 * <p>
 * @return dialogActive - boolean
 */
public boolean isDiaglogActive() {

	return dialogActive;
}


/**
 * Launches the help facility for this viewer's context.
 */
public void launchViewerContextHelp() {

	launchViewerContextHelpUsing("dataSourceDirectoryController");
}


/**
 * Creates a new Data Source Directory.
 * <p>
 * @return success - boolean
 */
public boolean newDirectory() {

	boolean completed = true;

	try {
		DataSourceDirectoryModel.getSessionModel().setNewModel();

		updateView();
	} catch (Exception exception) {
		fireDbResultsException(new
UnableToNewDataSourceDirectoryException(exception));
		completed = false;
	}

	return completed;
}

/**
```

```
	 * Opens a "saved" data source directory.
	 * <p>
	 * @return success - boolean
	 */
	public boolean openDirectory() {

		boolean completed = true;

		File file = getFileToOpen();

		try {
			if (file != null) {
				DataSourceDirectoryModel.getSessionModel().openModelIn(file);

				updateView();
			}
		} catch (Exception exception) {
			fireDbResultsException(new
UnableToOpenDataSourceDirectoryFileException(exception));
			completed = false;
		}

		return completed;
	}

	/**
	 * Pastes a directory node.
	 * <p>
	 * @return success - boolean
	 */
	public boolean pasteNode(ITreeFolderNodeModel folderNode) {

		boolean completed = true;

		try {

		getDataSourceDirectoryModel().pasteClipBoardNodeTo((DataSourceDirectoryFolderNo
deModel)getDataSourceDirectory().getFolderNodeOfSelectedDataSource());

			getDataSourceDirectory().invalidate();

			updateView();
		} catch (Exception exception) {
			fireDbResultsException(new
UnableToPasteDataSourceDirectoryElementException(exception));
			completed = false;
		}

		return completed;
	}

	/**
	 * Save As the data source directory.
	 * <p>
	 * @return success - boolean
	 */
	public boolean saveAsDirectory() {

		boolean completed = true;

		File file = getFileToSave();

		try {
			if (file != null)
				DataSourceDirectoryModel.getSessionModel().saveAs(file);
		} catch (Exception exception) {
			fireDbResultsException(new
UnableToSaveAsDataSourceDirectoryException(exception));
			completed = false;
		}

		return completed;
	}


	/**
```

```
     * Saves the data source directory.
     * <p>
     * @return success - boolean
     */
    public boolean saveDirectory() {

            boolean completed = true;

            try {
                    DataSourceDirectoryModel.getSessionModel().save();

                    updateView();
            } catch (Exception exception) {
                    fireDbResultsException(new
    UnableToSaveDataSourceDirectoryFileException(exception));
                    completed = false;
            }

            return completed;
    }

    /**
     * Sets that a dialog is active.
     */
    public void setDialogActive() {

            dialogActive = true;
    }


    /**
     * Sets if dataSources are to be previewed.
     * <p>
     * @return previewingOfConnections - boolean
     */
    public void setPreviewingOfDataSources(boolean mode) {

            previewingOfDataSources = mode;
    }

    /**
     * Sets if the Data Source is to be displayed.
     * <p>
     * @return showAsDbResults - boolean
     */
    public void setShowAsDbResults(boolean mode) {

            showAsDbResults = mode;
    }


    /**
     * Brings up a modal dialog with a specified icon, where the initial
     * choice is dermined by the <code>initialValue</code> parameter and
     * the number of choices is determined by the <code>optionType</code>
     * parameter.
     * <p>
     * If <code>optionType</code> is YES_NO_OPTION, or YES_NO_CANCEL_OPTION
     * and the <code>options</code> parameter is null, then the options are
     * supplied by the look and feel.
     * <p>
     * The <code>messageType</code> parameter is primarily used to supply
     * a default icon from the look and feel.
     *
     * @param parentComponent Determines the Frame in which the dialog is displayed.
     *                  If null, or if the parentComponent has no Frame, a
     *                  default Frame is used.
     * @param message   The Object to display
     * @param title     the title string for the dialog
     * @param optionType an int designating the options available on the dialog:
     *                   YES_NO_OPTION, or YES_NO_CANCEL_OPTION
     * @param messageType an int designating the kind of message this is,
     *                    primarily used to determine the icon from the pluggable
     *                    look and feel: ERROR_MESSAGE, INFORMATION_MESSAGE,
     *                    WARNING_MESSAGE, QUESTION_MESSAGE, or PLAIN_MESSAGE.
     * @param icon      the icon to display in the dialog
```

```
     * @param options   an array of objects indicating the possible choices
     *                  the user can make. If the objects are components, they
     *                  are rendered properly. Non-String objects are
     *                  rendered using their <code>toString</code> methods.
     *                  If this parameter is null, the options are determined
     *                  by the look and feel.
     * @param initialValue the object that represents the default selection
     *                     for the dialog
     * @return an int indicating the option chosen by the user,
     *         or CLOSED_OPTION if the user closed the Dialog
     */
    public VPDataSourceCustomizer showDataSourceWizardDialog(boolean isNewConnection,
    String dialogTitle) {

            //allow only one dialog occurence
            if(isDiaglogActive())
                    return null;

            setDialogActive();

            Beans.setDesignTime(true);

            VPDataSourceCustomizer customizer = new VPDataSourceCustomizer(this,
    isNewConnection);

            int choiceIndex = VCOptionDialog.showOptionDialog(
                    this,
            //parentComponent
                    customizer,                                                    //message -
    object to display
                    dialogTitle,                                          //title - title
    string for the dialog
                    VCOptionDialog.DEFAULT_OPTION,                //optionType -
    YES_NO_OPTION or YES_NO_CANCEL_OPTION
                    VCOptionDialog.QUESTION_MESSAGE,      //messageType - ERROR_MESSAGE,
    INFORMATION_MESSAGE, etc
                    null,                                                          //icon -
    icon to display in the dialog
                    DataSourceOptionNames,                        //options - array of
    possible choices
                    DataSourceOptionNames[1]);                    //initialValue - object
    that is the default selection

            Beans.setDesignTime(false);

            clearDialogActive();

            if(choiceIndex == VCOptionDialog.OK_OPTION)
                    return customizer;
            else
                    return null;
    }


    /**
     * Updates the datasource directory view.
     */
    public void updateView() {

            ((DataSourceDirectoryView)getView()).getDataSourceDirectory().updateView();
    }

    /**
     * Previews the database of the data source.
     * <p>
     * @param dataSource - the Data Source to preview
     */
    public void viewDataSource(MDbNodeTreeData dataSource) {

            traceMessage(getClass() + "  >>  viewDataBaseConnection");

            fireClearDesktopStatus();

            if (getPreviewingOfDataSources()) {
                    if (! getShowAsDbResults()) {
                            // just reuse exiting preview frame and preview listener
```

```
                        firePropertyChange("Connection Available", null, dataSource);
                } else {
                        // do not reuse preview frame plus getPreviewingOfConnections()
state does not matter
                        firePropertyChange("Connection Available", null, dataSource);
                }
        }
}
}
```

Right-click for Help

```
package com.dcr.dve.view.vcomponent.vcmenu;
/**
 * @(#)IVCMenuItem.java
 * <p>
 * ********************************************************************************
 * ********************************************************************************
 * <p>
 * IVCMenuItem provides a common interface for VCMenuItem components.
 * <p>
 * @author              Edward L. Stull
 *      @version 1.1
 * @since               JDK 2
 */
//345678901234567890123456789012345678901234567890*12345678901234567890123456789

import java.awt.event.ActionListener;
import java.awt.event.MouseEvent;

import javax.swing.Icon;
import javax.swing.MenuElement;
import javax.swing.MenuSelectionManager;

import com.dcr.dve.view.vcomponent.vccontrol.IVCCHelp;

public interface IVCMenuItem extends IVCCHelp {
/**
 * Adds an Action Listener
 * <p>
 * @param listener - ActionListener
 */
public void addActionListener(ActionListener listener);


/**
 * Initializes the menu item and register the result with the menuitem
 *   hashtable so that it can be fetched with getMenuItem().
 * <p>
 * @see #getMenuItem
 */
public void initialize();


/**
 * Overrides the processing of the event modifiers.
 * <p>
 * Process a mouse event. event is a MouseEvent with source being the receiving
component.
 *   componentPath is the path of the receiving MenuElement in the menu
 *   hierarchy. manager is the MenuSelectionManager for the menu hierarchy.
 * This method should process the MouseEvent and change the menu selection if
necessary
 *   by using MenuSelectionManager's API.
 * <p>
 * Note: you do not have to forward the event to sub-components. This is done
automatically
 *   by the MenuSelectionManager
 */
public void processMouseEvent(MouseEvent event, MenuElement path[],
MenuSelectionManager manager);


/**
 * Sets the Action Command
 * <p>
 * @param commandName java.lang.String
 */
public void setActionCommand(String commandName);


/**
 * Sets the enable mode.
 * <p>
 * @param mode boolean
 */
```

```
public void setEnabled(boolean mode);


/**
 * Sets the horizontal text position
 * <p>
 * @param alignment - int
 */
public void setHorizontalTextPosition(int alignment);


/**
 * Sets the icon
 * <p>
 * @param icon - Icon
 */
public void setIcon(Icon icon);


/**
 * Sets the mnemonic.
 * <p>
 * @param keyAccelerator - int (char)
 */
public void setMnemonic(int keyAccelerator);


/**
 * Sets the text.
 * <p>
 * @param text java.lang.String
 */
public void setText(String text);
}


package com.dcr.dve.view.vcomponent.vccontrol;
/**
 * @(#)IVCCHelp.java
 * <p>
 * ********************************************************************************
 * ********************************************************************************
 * <p>
 * IVCCHelp provides a common interface for help-related components.
 * <p>
 * @author              Edward L. Stull (ID: ELS)
 *      @version 2.006
 * @since               JDK 2
 */
//3456789012345678901234567890123456789012345678901234567890123456789*123456789012345678901234567890

import java.awt.event.MouseEvent;

import javax.swing.MenuElement;
import javax.swing.MenuSelectionManager;

public interface IVCCHelp {
/**
 * Gets the name of the command.
 * <p>
 * @return name java.lang.String
 */
public String getCommandName();


/**
 * Gets the default name of the command.
 * <p>
 * @return name java.lang.String
 */
public String getDefaultName();


/**
 * Gets the string associated with the resource tag.
 * <p>
```

```
 * @param name java.lang.String
 * @return name java.lang.String
 */
public String getResourceString(String resourceTagName);
}


package com.dcr.dve.view.vcomponent.vccontrol;
/**
 * @(#)VCControlKit.java
 * <p>
* ********************************************************************************
 * ********************************************************************************
 * <p>
 * An <code>VCControlBar</code> is a set of helper methods supporting
 *  the operation of controls, that is, active components. Currently, this
 *  kit provides context help support in menu and toolbar controls.
 * <p>
 * @author              Edward L. Stull
 * @version 2.11
 * @since               JDK 2
 */
//345678901234567890123456789012345678901234567890123456789*12345678901234567890123456789

import java.awt.event.MouseEvent;

import javax.swing.MenuElement;
import javax.swing.MenuSelectionManager;

import com.dcr.dve.view.vcomponent.vchtml.VCDefaultBrowser;

public class VCControlKit extends Object {
/**
 * Gets the URL text for the contextual help name.
 * <p>
 * @param helpTag - a resource tag for help
 * @return contextHelpUrlName
 */
public static String getContextHelpUrlLabelFor(String helpTag) {

        return helpTag + getContextHelpUrlLabelSuffix();
}


/**
 * Gets the label for the contextual help URL.
 * <p>
 * @return contextHelpUrlLabel
 */
public static String getContextHelpUrlLabelSuffix() {

        return "HelpUrl";
}


/**
 * Gets the URL text for the contextual help name using a view that implements
 *   the help interface.
 * <p>
 * @param helpview - a view that implements the help interface
 * @return contextHelpUrlText
 */
public static String getContextHelpUrlTextFor(IVCCHelp helpView) {

        return helpView.getResourceString(helpView.getCommandName() +
getContextHelpUrlLabelSuffix());
}


/**
 * Answer if this is a help context event.
 * <p>
 * @param event - the mouse event.
 * @return boolean
 * @see         java.awt.event.MouseEvent
 * @see         java.awt.event.MouseListener
```

```
 * @see         java.awt.Component#addMouseListener
 * @see         java.awt.Component#enableEvents
 */
public static boolean isContextHelpEvent(MouseEvent event) {

        return ((event.getID() == MouseEvent.MOUSE_RELEASED)
                        && ((event.getModifiers() & MouseEvent.META_MASK) != 0));
}


/**
 * Launch the help facility for a view that implements
 *   the help interface.
 * <p>
 * @param helpview - a view that implements the help interface
 */
public static void launchViewerContextHelpUsing(IVCCHelp helpView) {

        String defaultHelpUrlText = getContextHelpUrlTextFor(helpView);

        String fileSep = System.getProperty("file.separator"); // e.g., "/"

        String helpUrlPath = System.getProperty("user.dir") + fileSep + "Help" +
fileSep;

        VCDefaultBrowser browser = new VCDefaultBrowser();
        if (defaultHelpUrlText == null)
                browser.displayURL(helpUrlPath + "Action" + fileSep +
helpView.getDefaultName().replace(' ', '_') + ".htm");
        else
                browser.displayURL(helpUrlPath + defaultHelpUrlText);
}

/**
 * Launch the help facility for this viewer's context using a help URL text.
 * <p>
 * @param helpUrlText - String
 */
public static void launchViewerContextHelpUsing(String helpUrlText) {

        if (helpUrlText == null)
                helpUrlText = "index.htm";

        String fileSep = System.getProperty("file.separator"); // e.g., "/"

        VCDefaultBrowser browser = new VCDefaultBrowser();
        browser.displayURL(System.getProperty("user.dir") + fileSep + "Help" + fileSep
+ helpUrlText);
}
}


package com.dcr.dve.view.vcomponent.vctool;
/**
 * @(#)VCToolBarButton.java
 * <p>
 * ********************************************************************************
 * ********************************************************************************
 * <p>
 * An implementation of a button for the viewer toolbars.
 * <p>
 * @author              Edward L. Stull
 *      @version 2.009
 * @since               JDK 2
 */
//345678901234567890123456789012345678901234567890+12345678901234567890123456789

import java.awt.Insets;
import java.awt.event.MouseEvent;
import java.net.URL;
import java.util.Hashtable;
import java.util.ResourceBundle;

import javax.swing.ImageIcon;
import javax.swing.border.Border;
import javax.swing.border.CompoundBorder;
```

```
import javax.swing.border.EmptyBorder;
import javax.swing.border.EtchedBorder;

import com.dcr.dve.view.vcomponent.VCAction;
import com.dcr.dve.view.vcomponent.VCPushButton;
import com.dcr.dve.view.vcomponent.vcbutton.VCButton;
import com.dcr.dve.view.vcomponent.vccontrol.IVCCHelp;
import com.dcr.dve.view.vcomponent.vccontrol.VCControlKit;
import com.dcr.dve.view.vcomponent.vchtml.VCDefaultBrowser;

import com.dcr.dvg.model.ResourcesKit;

public class VCToolBarButton extends VCPushButton implements IVCCHelp {

        protected VCToolBar controller = null; // the controller for this component
        protected String defaultName;
        protected String tagName; // name used for tag in the resource
        protected ImageIcon imageIcon;

        protected final static Border emptyBorder = new EmptyBorder(2, 2, 2, 2);
        protected final static Border etchedBorder
          = new CompoundBorder(new EtchedBorder(), emptyBorder);

/*
 * Constructor.
 * <p>
 * @param controller - the tool bar controller
 * @param tagName name used for tag in the resource
 */
public VCToolBarButton(VCToolBar controller, String tagName) {

        super();

        initialize(controller, tagName);
}


/**
 * Launchs the help facility for this viewer's context.
 */
public void contextHelpAction() {

        VCControlKit.launchViewerContextHelpUsing(this);
}

/**
 * Gets the action for the specified command.
 * <p>
 * @param      String - command
 */
protected VCAction getAction(String command) {

        return (VCAction)getCommands().get(command);
}


/**
 * Gets the action suffix string.
 * <p>
 * @return image - suffix
 */
public String getActionSuffix() {

        return getController().getActionSuffix();
}


/**
 * Gets Y alignment.
 * <p>
 * @return alignment
 */
public float getAlignmentY() {

        return 0.5f;
}
```

```
/**
 * Gets the action name for the menu item as specified in the resource.
 * <p>
 * @param       String- command
 */
protected VCAction getButtonAction(String commandName) {

        String actionName = getResourceString(commandName + getActionSuffix());

        if (actionName == null)
                actionName = commandName; //defalut action name to that of the command
name

        return getAction(actionName);
}


/**
 * Gets the name (that is, tag name) for the buttom that is used by the resource.
 * <p>
 * @return tagName - java.lang.String
 */
public String getCommandName() {

        return tagName;
}


/**
 * Gets the commands of the controller.
 * <p>
 * @return  commands - java.util.Hashtable;
 */
public Hashtable getCommands() {

        return getController().getCommands();
}


/**
 * Gets the controller for this resource-employing component.
 * <p>
 * @return controller - the VCToolBar controller
 */
public VCToolBar getController() {

        return controller;
}


/**
 * Gets the name for the buttom that is used by the resource.
 * <p>
 * @return defaultName - String
 */
public String getDefaultName() {

        return defaultName;
}


/**
 * Gets the image(icon) based on the image name.
 * <p>
 * @param imageName java.lang.String
 * @return ImageIcon
 */
public ImageIcon getImageIcon(String imageName) {

        return getController().getImageIcon(imageName);
}


/**
```

```
     * Gets the image suffix string.
     * <p>
     * @return imageSuffix - String
     */
    public String getImageSuffix() {

            return getController().getImageSuffix();
    }


    /**
     * Gets the label suffix string.
     * <p>
     * @return labelSuffix - String
     */
    public String getLabelSuffix() {

            return getController().getLabelSuffix();
    }


    /**
     * Gets a resource from the local resource or from a super class resource.
     * <p>
     * @param key
     * @return URL
     */
    public URL getResource(String key) {

            return getController().getResource(key);
    }


    /**
     * Gets the resources.
     * <p>
     * @return ResourceBundle
     */
    public ResourceBundle getResources() {

            return getController().getResources();
    }


    /**
     * Gets a resource string from the local resource or from a super class resource.
     * <p>
     * @param key java.lang.String
     * @return java.lang.String
     */
    public String getResourceString(String key) {

            return getController().getResourceString(key);
    }


    /**
     * Gets the image suffix string.
     * <p>
     * @return tipSuffix - String
     */
    public String getTipSuffix() {

            return getController().getTipSuffix();
    }


    /**
     * Initializes the menu item and register the result with the menuitem
     *   hashtable so that it can be fetched with getMenuItem().
     * <p>
     * @param controller
     * @param tagName name used for tag in the resource
     * @see #getMenuItem
     */
    public void initialize(VCToolBar controller, String tagName) {
```

```
        setController(controller);

        setUI();
        setCommandName(tagName);
        setLabelFromResource();
        setCrossActionAndButtonRegistration();
        setImageFromResource();
        setToolTipText();
}


/**
 * Answer if the icon is to be shown.
 */
public boolean isShowIcon() {

        return true;
}


/**
 * Override to process the event modifiers.
 * <p>
 * Processes mouse events occurring on this component by
 * dispatching them to any registered
 * <code>MouseListener</code> objects.
 * <p>
 * This method is not called unless mouse events are
 * enabled for this component. Mouse events are enabled
 * when one of the following occurs:
 * <p><ul>
 * <li>A <code>MouseListener</code> object is registered
 * via <code>addMouseListener</code>.
 * <li>Mouse events are enabled via <code>enableEvents</code>.
 * </ul>
 * @param       event - the mouse event.
 * @see         java.awt.event.MouseEvent
 * @see         java.awt.event.MouseListener
 * @see         java.awt.Component#addMouseListener
 * @see         java.awt.Component#enableEvents
 * @since       JDK1.1
 */
public void processMouseEvent(MouseEvent event) {

        if (VCControlKit.isContextHelpEvent(event))
                VCControlKit.launchViewerContextHelpUsing(this);
                //((MouseListener)mouseListener.a).model.setPressed(false);
                //mouseListener.model.setPressed(false);
        else
                super.processMouseEvent(event);
}


/**
 * Sets the name for the buttom that is used by the resource.
 * <p>
 * @param tagName
 */
public void setCommandName(String tagName) {

        this.tagName = tagName;
}


/**
 * Sets the controller for this resource-employing component.
 * <p>
 * @param controller - the VCToolBar controller
 */
public void setController(VCToolBar controller) {

        this.controller = controller;
}
```

```
/**
 * Cross registers the action and menu item. Also, map
 *   the command name to this menu item.
 */
public void setCrossActionAndButtonRegistration() {

        String commandName = getCommandName();

        VCAction action = getButtonAction(commandName);

        if (action == null)
                action = getButtonAction(getDefaultName());

        if (action != null) {
                // set the action commnand name that is included in the event sent to
action listeners
                setActionCommand((String)action.getValue(action.NAME));
                addActionListener(action);
                setEnabled(action.isEnabled());
                setIcon(action.getIcon());

                action.addPropertyChangeListener(new VCToolBarButtonListener(this));
        } else {
                setEnabled(false);
                /*debug trace*///System.err.println("Action for button \"" +
commandName + "\" not resolved.");
                ResourcesKit.fireResourceTagError(getResources(), getDefaultName());
        }
}


/**
 * Sets the default name for the buttom.
 * <p>
 * @param defaultName
 */
public void setDefaultName(String defaultName) {

        this.defaultName = defaultName;
}


/**
 * Sets the button's image.
 */
public void setImageFromResource() {

        if (getIcon() == null) { // if not specified by the action
                String commandName = getCommandName();
                URL url = getResource(commandName + getImageSuffix());
                if (url != null) {
                        if (isShowIcon())
                                setHorizontalTextPosition(VCButton.CENTER);
                        else
                                setHorizontalTextPosition(VCButton.LEFT);

                        setIcon(new
ImageIcon(java.awt.Toolkit.getDefaultToolkit().getImage(url)));
                } else

                        // default to image with the command name
                        setIcon(getImageIcon(getDefaultName()));
        }
}


/**
 * Sets the menu item's label in the menu list.
 * <p>
 * @param command name
 */
protected void setLabelFromResource() {

        String commandName = getCommandName();
        String label = getResourceString(commandName + getLabelSuffix());
```

```
        //defalut the label
        if (label == null)
                label = getResourceString(commandName + getActionSuffix());
        if (label == null)
                label = commandName;

        //setText(label);

        setDefaultName(label);
}


/**
 * Answer if the icon is to be shown.
 */
public boolean setShowIcon() {

        return true;
}


/**
 * For tool bar buttons, override text set to only set the name
 * and the tool tip.
 * <p>
 * @param buttonText - String - the text to display with the button
 */
public void setText(String buttonText) {

        setDefaultName(buttonText);
        setToolTipText(buttonText);
}


/**
 * Sets the tool tip text based on the resource specification.
 */
protected void setToolTipText() {

        String tip = getResourceString(getCommandName() + getTipSuffix());
        if (tip != null)
                super.setToolTipText(tip);
        else
                super.setToolTipText(getDefaultName());
}


/**
 * Sets UI for this component.
 */
public void setUI() {

        setMargin(new Insets(1,1,1,1));

        Border emptyBorder = new EmptyBorder(1,1,1,1);
        setBorder(emptyBorder);

        setRequestFocusEnabled(false);
}
}
```

Transcripts

```
package com.dcr.dvg.model.transcript;
/**
 * @(#)Transcript.java
 * <p>
 * ********************************************************************************
* ********************************************************************************
 * <p>
 * The class <code>Transcript</code> and its subclasses are models of
 *   logged events that potentially can be edited.
 * <p>
 * @author              Edward L. Stull
 * @version 1.0.2
 * @since               JDK 2
 */
//345678901234567890123456789012345678901234567890123456789*12345678901234567890123456789

import java.util.Date;

import javax.swing.text.DefaultStyledDocument;
import javax.swing.text.StyleContext;

import com.dcr.dve.model.muser.MUserContext;

import com.dcr.dvg.model.DvgProductInfo;

public class Transcript extends DefaultStyledDocument {

        protected String dvgVersion = null;
        protected String userId = null;
        protected Date creationDate = null;
        protected Date lastEditedDate = null;

/**
 * Transcript constructor.
 */
public Transcript() {

        this(new StyleContext());
}


/**
 * Constructor based on a set of styles.
 * <p>
 * @param styles - StyleContext
 */
public Transcript(StyleContext styles) {

        super(styles);

        setUserId(MUserContext.getUserId());
        setDvgVersion(DvgProductInfo.getDvgVersion());
        setCreationDate(new Date());
        setLastEditedDate(new Date());
}


/**
 * Gets the creation date.
 * <p>
 * @return creationDate  Date
 */
public Date getCreationDate() {

        return creationDate;
}


/**
 * Gets the version.
 * <p>
 * @return version java.lang.String
 */
public String getDvgVersion() {
```

```
        return dvgVersion;
}


/**
 * Gets the date of the last edit.
 * <p>
 * @return java.util.Date
 */
public Date getLastEditedDate() {

        return lastEditedDate;
}


/**
 * Gets the user's ID.
 * <p>
 * @return userId - java.lang.String
 */
public String getUserId() {

        return userId;
}


/**
 * Sets the creation date.
 * <p>
 * @param creationDate java.util.Date
 */
private void setCreationDate(Date creationDate) {

        this.creationDate = creationDate;
}


/**
 * Sets the version.
 * <p>
 * @param dvgVersion java.lang.String
 */
private void setDvgVersion(String dvgVersion) {

        this.dvgVersion = dvgVersion;
}


/**
 * Sets the date of the last edit.
 * <p>
 * @param lastEditedDate java.util.Date
 */
private void setLastEditedDate(Date lastEditedDate) {

        this.lastEditedDate = lastEditedDate;
}


/**
 * Sets the user's ID.
 * <p>
 * @param userId java.lang.String
 */
private void setUserId(String userId) {

        this.userId = userId;
}
}


package com.dcr.dvg.model.transcript;
/**
 * @(#)MasterTranscript.java
 * <p>
```

```
 * ********************************************************************************
 * ********************************************************************************
 * <p>
 * The class <code>MasterTranscript</code> is a read-only model of logged events.
 *   It is opened, updated and saved under system control. It cannot be edited by
 *   a user.
 * <p>
 * @author              Edward L. Stull
 * @version 1.2
 * @since               2
 */
//345678901234567890123456789012345678901234567890*12345678901234567890123456789

import javax.swing.text.StyleContext;

public class MasterTranscript extends Transcript {
/**
 * Constructor.
 */
public MasterTranscript() {

        super();
}


/**
 * Constructor.
 * <p>
 * @param styles javax.swing.text.StyleContext
 */
public MasterTranscript(StyleContext styles) {

        super(styles);
}
}
package com.dcr.dvg.view.controller.transcript;
/**
 * @(#)TranscriptController.java
 * <p>
 * ********************************************************************************
 * ********************************************************************************
 * <p>
 * The class <code>TranscriptController</code> controls, that is manages,
 *   all transcripts, one of which is a read-only master transcript and
 *   the other are optionally-created user-editable transcripts.
 * <p>
 * @author              Edward L. Stull
 * @version 1.8
 * @since               JDK 2
 */
//345678901234567890123456789012345678901234567890*12345678901234567890123456789

import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.beans.PropertyChangeEvent;
import java.io.IOException;
import java.util.Date;

import javax.swing.Action;
import javax.swing.JLayeredPane;
import javax.swing.text.JTextComponent;
import javax.swing.text.TextAction;

import com.dcr.dve.model.mdb.MDbNodeTreeData;
import com.dcr.dve.model.mcommand.mccompiler.mccparser.MCCCommand;
import com.dcr.dvg.model.transcript.exception.CannotCloseMasterTranscriptException;
import
com.dcr.dvg.model.transcript.exception.CannotSaveAPreviousMasterTranscriptException;
import com.dcr.dvg.model.transcript.exception.CannotSaveAsMasterTranscriptException;
import com.dcr.dvg.util.throwable.DVError;
import com.dcr.dvg.util.throwable.DVException;
import com.dcr.dvg.util.throwable.NoDataSourceSelectedForQueryExecutionException;
import com.dcr.dve.view.vcomponent.vctext.IVCEditor;
import com.dcr.dve.view.vcomponent.vctext.VCRichTextEditor;
import com.dcr.dve.view.vcomponent.vctext.VCRichTextParagraph;
import com.dcr.dve.view.vcomponent.vctext.VCRichTextRun;
```

```
import com.dcr.dve.view.vcomponent.vctext.VCRichTextRunList;
import com.dcr.dve.view.vprocess.VPRichTextEditorViewer;

import com.dcr.dve.view.vprocess.vpauxiliaryaction.VPAuxiliaryGeneralHelpAction;
import
com.dcr.dve.view.vprocess.vpcommand.vpcaction.VPCommandTranscriptControllerHelpAction;

import com.dcr.dve.view.vprocess.vpcommand.vpcaction.VPCommandExecuteSelectionAction;
import com.dcr.dve.view.vprocess.vpcommand.vpcaction.VPCommandExecutePreviousAction;

import com.dcr.dve.view.vprocess.vpcommand.vpcaction.VPCommandClearContextAction;
import
com.dcr.dve.view.vprocess.vpcommand.vpcaction.VPCommandToggleAutoInsertContextAction;
import
com.dcr.dve.view.vprocess.vpcommand.vpcaction.VPCommandToggleShowCommandParseAction;
import
com.dcr.dve.view.vprocess.vpcommand.vpcaction.VPCommandToggleShowLexerParseAction;
import
com.dcr.dve.view.vprocess.vpcommand.vpcaction.VPCommandToggleSuggestCommandContextActi
on;
import
com.dcr.dve.view.vprocess.vpcommand.vpcaction.VPCommandToggleValueFillContextAction;
import
com.dcr.dve.view.vprocess.vpcommand.vpcaction.VPCommandToggleValuePaddedContextAction;

import com.dcr.dvg.model.transcript.MasterTranscript;
import com.dcr.dvg.model.transcript.Transcript;
import com.dcr.dvg.model.transcript.exception.CannotCloseMasterTranscriptException;
import
com.dcr.dvg.model.transcript.exception.CannotSaveAPreviousMasterTranscriptException;
import com.dcr.dvg.model.transcript.exception.CannotSaveAsMasterTranscriptException;
import com.dcr.dvg.view.component.desktop.DVGDesktop;
import com.dcr.dvg.view.component.text.rich.IRichTextFileEditorEmbedded;
import com.dcr.dvg.view.controller.datasource.DataSourceDirectory;
import com.dcr.dvg.view.controller.datasource.DataSourceDirectoryControllerFrame;
import com.dcr.dvg.view.controller.iteration.IterationControllerFrame;
import com.dcr.dvg.view.desktop.VPDesktopViewer;
import com.dcr.dvg.view.desktop.DVGDesktopViewer;
import com.dcr.dvg.view.desktop.StatusViewer;

public class TranscriptController extends VPRichTextEditorViewer {

        protected transient JLayeredPane desktopView = null;

        protected static String previousExecutedCommand = null;

        public static final String CLEAR_DESKTOP_STATUS = "Clear Desktop Status";
        public static final String OPERATION_BEGUN = "Operation Begun";
        public static final String OPERATION_CANCELED = "Operation Canceled";
        public static final String OPERATION_FAILED = "Operation Failed";
        public static final String OPERATION_FINISHED = "Operation Finished";
        public static final String POST_ERROR = "Post Error";
        public static final String POST_STATUS = "Post Status";
        public static final String POST_WARNING = "Post Warning";

/**
 * Constructor.
 *              FOR TESTING ONLY
 *              FOR TESTING ONLY
 *              FOR TESTING ONLY
 *              FOR TESTING ONLY
 */
public TranscriptController() {

        super();

        setDesktopView(new DVGDesktop());
}


/**
 * Constructor based on the desktop.
 */
public TranscriptController(JLayeredPane desktopView) {

        super();
```

```
        setDesktopView(desktopView);

        initialize();
}


/**
 * Clears the desktop status.
 */
public void clearDesktopStatus() {

        getDesktopStatusViewer().clearStatus();
}


/**
 * Close.
 */
public void close()
  throws DVException {

        //do nothing
}

/**
 * Closes the active transcript.
 */
public void closeAction() {

        try {
                getTranscriptTabsViewer().closeTranscript();
        } catch (IOException exception1) {
                postWarningStatus(exception1.toString());
        } catch (CannotCloseMasterTranscriptException exception2) {
                postWarningStatus(exception2.toString());
        }

        refresh();
}

/**
 * Launches a command execution of the command text.
 * <p>
 * Synchronized to maintain order of printing to the transcript.
 * <p>
 * @param commandText - String
 */
public synchronized void executeCommand(String commandText) {

        traceMessage("starting parse with PREVIOUS commands=" + commandText);

        postOperationBegun("Executing " + commandText);

        MDbNodeTreeData selectedConnection = null;
        try {
                selectedConnection = getSelectedDataSource();
        } catch (DVException exception) {
                fireDVException(exception);
        }

        if (selectedConnection == null) {
                postDbResultsException(new
NoDataSourceSelectedForQueryExecutionException());
        } else {
                MCCCommand newParser = new MCCCommand();

                // NOT YET IMPLEMENTED BELOW
                // frame is added to the desktop later
//              VPDesktopFrame newFrame = getDesktopViewer().getNewResultsFrame();
//
//              newParser.addPropertyChangeListener(newFrame);
//              newParser.addPropertyChangeListener(this);
//              newFrame.addPropertyChangeListener(newParser);
//
//              newFrame.fireExecuteCommands(commandText);
```

```
//              newFrame.refresh();
                // NOT YET IMPLEMENTED ABOVE
        }

        postOperationFinished("Executing " + commandText);
}


/**
 * Launches a command execution of the previous command.
 */
public void executePrevious() {

        String commandText = getPreviousExecutedCommand();

        if (commandText == null)
                commandText = getDefaultExecutionText();

        executeCommand(commandText);
}


/**
 * Launches a command execution of the selected text.
 */
public void executeSelection() {

        String commandText = ((JTextComponent)getEditor()).getSelectedText();

        traceMessage("starting parse with selection=" + commandText);

        setPreviousExecutedCommand(commandText);

        executePrevious();
}


/**
 * Gets the list of actions supported by this
 * editor.  It is implemented to return the list
 * of actions supported by the embedded JTextComponent
 * augmented with the actions defined locally.
 */
public Action[] getActions() {

        Action[] defaultActions = {
                new VPCommandExecuteSelectionAction(this),
                new VPCommandExecutePreviousAction(this),

                new VPCommandClearContextAction(this),
                new VPCommandToggleAutoInsertContextAction(this),
                new VPCommandToggleSuggestCommandContextAction(this),
                new VPCommandToggleValueFillContextAction(this),
                new VPCommandToggleValuePaddedContextAction(this),

                new VPCommandToggleShowCommandParseAction(this),
                new VPCommandToggleShowLexerParseAction(this),

                new VPAuxiliaryGeneralHelpAction(this),
                new VPCommandTranscriptControllerHelpAction(this)
        };

        return TextAction.augmentList(super.getActions(), defaultActions);
}


/**
 * Gets the active (i.e., one being edited) transcript viewer.
 * <p>
 * @param transcriptViewer - TranscriptViewer
 */
public TranscriptViewer getActiveTranscriptViewer() {

        return  ((TranscriptTabsViewer)getView()).getActiveTranscriptViewer();
}
```

```
/**
 * Gets the Data Source Directory (view).
 * <p>
 * @param dataSourceDirectory - DataSourceDirectory
 */
public DataSourceDirectory getDataSourceDirectory() {

        return (DataSourceDirectory)
                        ((DataSourceDirectoryControllerFrame)
                          getDesktopViewer().getDataSourceDirectoryControllerFrame())
                        .getTypedViewer().getDataSourceDirectory();
}

/**
 * Gets the default execution text.
 * <p>
 * @return defaultExecutionText
 */
protected String getDefaultExecutionText() {

        return getClass() + ">>getDefaultExecutionText";
}


/**
 * Gets the desktop status viewer.
 * <p>
 * @return statusViewer - StatusViewer
 */
public StatusViewer getDesktopStatusViewer() {

        return
          ((DVGDesktopViewer)
            ((DVGDesktop)getDesktopView()).getController()).
                  getDesktopStatusViewer();
}


/**
 * Gets the desk top view.
 * <p>
 * @return desktopView - JLayeredPane
 */
public JLayeredPane getDesktopView() {

        return desktopView;
}


/**
 * Gets the desktop viewer.
 * <p>
 * @return desktopViewer - DVGDesktopViewer
 */
public DVGDesktopViewer getDesktopViewer() {

        return (DVGDesktopViewer)((DVGDesktop)getDesktopView()).getController();
}


/**
 * This a standard interface to access the transcript.
 * <p>
 * Gets the editor.
 * <p>
 * @return editor - IVCEditor
 */
public IVCEditor getEditor() {

        return (IVCEditor)((TranscriptTabsViewer)getView()).getEditor();
}


/**
 * Gets the iteration control frame.
 * <p>
```

```
 * @return controllerFrame - IterationControllerFrame
 */
public IterationControllerFrame getIterationControllerFrame() {

        return ((DVGDesktopViewer)

        ((DVGDesktop)getDesktopView()).getController()).getIterationControllerFrame();
}


/**
 * Gets the previous executed command.
 * <p>
 * @return previousExecutedCommand - String
 */
public String getPreviousExecutedCommand() {

        return previousExecutedCommand;
}


/**
 * Gets the selected Data Source from the Data Source directory.
 * <p>
 * @param selectedDataSource - MDbNodeTreeData
 */
public MDbNodeTreeData getSelectedDataSource()
  throws DVException {

        return (MDbNodeTreeData)getDataSourceDirectory().getSelectedDataSource();
}


/**
 * Gets the transcript at a designated index in the tab viewer.
 * <p>
 * @return masterTranscript TranscriptViewer
 */
public TranscriptViewer getTranscriptAt(int index) {

        return ((TranscriptTabsViewer)getView()).getTranscriptAt(index);
}


/**
 * Gets the master transcript.
 * <p>
 * @return masterTranscript VPCommandTranscriptBasic
 */
public int getTranscriptCount() {

        return ((TranscriptTabsViewer)getView()).getTranscriptCount();
}


/**
 * Gets the transcript tabs viewer.
 * <p>
 * @return tabsViewer - TranscriptTabsViewer
 */
public TranscriptTabsViewer getTranscriptTabsViewer() {

        return (TranscriptTabsViewer)getView();
}

/**
 * Initializes this viewer.
 */
public boolean initialize() {

        initialize("TranscriptController", new TranscriptTabsViewer(this));

        getEditor().addPropertyChangeListener(this);

        return true;
}
```

```
/**
 * Answer true if the active transcript is a master transcript.
 * <p>
 * @return answer - boolean
 */
public boolean isAMasterTranscript() {

        return ((TranscriptTabsViewer)getView()).isAMasterTranscript();
}

/**
 * Launches the help facility for this viewer's context.
 */
public void launchViewerContextHelp() {

        launchViewerContextHelpUsing("transcriptController");
}


/**
 * Creates and opens a new active transcript and file.
 */
public void newAction() {

        getTranscriptTabsViewer().newTranscript();

        revalidate();
}

/**
 * Opens and makes active an existing transcript and file.
 */
public void openAction() {

        getTranscriptTabsViewer().openTranscript();

        revalidate();
}

/**
 * Pastes rich text paragraph to the transcripts.
 * <p>
 * @param richTextParagraph - VCRichTextParagraph
 */
public void pasteRichTextToTranscript(VCRichTextParagraph richTextParagraph) {

        getTranscriptTabsViewer().pasteRichTextToTranscript(richTextParagraph);
}

/**
 * Pastes a VCRichTextParagraph to the transcripts.
 * <p>
 * @param text - String
 */
public void pasteToTranscript(VCRichTextParagraph richTextParagraph) {

        traceMessage(getClass() + "  >>  pasteToTranscript=" + richTextParagraph);

        pasteRichTextToTranscript(richTextParagraph);
}


/**
 * Pastes an error to the transcripts.
 * <p>
 * @param error - DVError
 */
public void pasteToTranscript(DVError error) {

        traceMessage(getClass() + "  >>  pasteToTranscript=" + error);

        pasteRichTextToTranscript(error.toRichTextParagraph());

        repaint();
```

```
}

/**
 * Pastes an exception to the transcripts.
 * <p>
 * @param exception - DVException
 */
public void pasteToTranscript(DVException exception) {

        traceMessage(getClass() + "  >>  pasteToTranscript=" + exception);

        pasteRichTextToTranscript(exception.toRichTextParagraph());

        repaint();
}


/**
 * Pastes text to the transcripts.
 * <p>
 * @param text - String
 */
public void pasteToTranscript(String text) {

        traceMessage(getClass() + "  >>  pasteToTranscript=" + text);

        VCRichTextRunList commandRunList = new VCRichTextRunList();
        commandRunList.addFirst(VCRichTextRun.KEYWORD, text);
        VCRichTextParagraph commandRunParagraph = new VCRichTextParagraph("command",
commandRunList);

        pasteRichTextToTranscript(commandRunParagraph);
}


/**
 * Posts "DB Results Exception" property change.
 * <p>
 * Localize this exception to this viewer.
 * <p>
 * @param exception - DVException
 */
public void postDbResultsException(DVException exception) {

        postStatus(exception);
}

/**
 * Posts an error property change.
 * <p>
 * @param errorMessage - DVException
 */
public void postErrorStatus(String errorMessage) {

        getDesktopStatusViewer().postErrorStatus(errorMessage);

        pasteToTranscript(errorMessage);
}


/**
 * Posts the logoff operation.
 */
public void postLogoff() {

        clearDesktopStatus();

        VCRichTextRunList commandRunList = new VCRichTextRunList();
        commandRunList.addLast(VCRichTextRun.HEADING, "Logoff at ");
        commandRunList.addLast(VCRichTextRun.HEADING, new Date().toString());

        pasteRichTextToTranscript(new VCRichTextParagraph("command", commandRunList));

        getTranscriptTabsViewer().close();
}
```

```
/**
 * Posts the logon operation.
 */
public void postLogon(String userNameText) {

        clearDesktopStatus();

        VCRichTextRunList commandRunList = new VCRichTextRunList();
        commandRunList.addLast(VCRichTextRun.HEADING, "Logon at ");
        commandRunList.addLast(VCRichTextRun.HEADING, new Date().toString());
        commandRunList.addLast(VCRichTextRun.HEADING, " by ");
        commandRunList.addLast(VCRichTextRun.EMPHASIZEDKEYWORD, ":  " + userNameText);

        pasteRichTextToTranscript(new VCRichTextParagraph("command", commandRunList));
}


/**
 * Posts the beginning of an operation.
 */
public void postOperationBegun(String operationName) {

        clearDesktopStatus();

        VCRichTextRunList commandRunList = new VCRichTextRunList();
        commandRunList.addLast(VCRichTextRun.HEADING, "Operation begun on ");
        commandRunList.addLast(VCRichTextRun.HEADING, new Date().toString());
        commandRunList.addLast(VCRichTextRun.EMPHASIZEDKEYWORD, ":  " + operationName);

        pasteRichTextToTranscript(new VCRichTextParagraph("command", commandRunList));
}


/**
 * Posts the cancel of an operation.
 * <p>
 * @param operationName - String - the name of the operation
 */
public void postOperationCanceled(String operationName) {

        getDesktopStatusViewer().postWarningStatus("Operation CANCELED:  " +
operationName);

        VCRichTextRunList commandRunList = new VCRichTextRunList();
        commandRunList.addLast(VCRichTextRun.EMPHASIZEDKEYWORD, "Operation CANCELED on
");
        commandRunList.addLast(VCRichTextRun.EMPHASIZEDKEYWORD, new Date().toString());
        commandRunList.addLast(VCRichTextRun.EMPHASIZEDKEYWORD, ":  " + operationName);

        pasteRichTextToTranscript(new VCRichTextParagraph("command", commandRunList));
}


/**
 * Posts the failure of an operation.
 * <p>
 * @param operationName - String - the name of the operation
 */
public void postOperationFailed(String operationName) {

        getDesktopStatusViewer().postWarningStatus("Operation FAILED:  " +
operationName);

        VCRichTextRunList commandRunList = new VCRichTextRunList();
        commandRunList.addLast(VCRichTextRun.ERROR, "Operation >>>>> FAILED <<<<< on
");
        commandRunList.addLast(VCRichTextRun.ERROR, new Date().toString());
        commandRunList.addLast(VCRichTextRun.ERROR, ":  " + operationName);

        pasteRichTextToTranscript(new VCRichTextParagraph("command", commandRunList));
}


/**
```

```
 * Posts the finish of an operation.
 * <p>
 * @param operationName - String - the name of the operation
 */
public void postOperationFinished(String operationName) {

        VCRichTextRunList commandRunList = new VCRichTextRunList();
        commandRunList.addLast(VCRichTextRun.EMPHASIZEDKEYWORD, "Operation finished on
");
        commandRunList.addLast(VCRichTextRun.EMPHASIZEDKEYWORD, new Date().toString());
        commandRunList.addLast(VCRichTextRun.EMPHASIZEDKEYWORD, ":  " + operationName);

        pasteRichTextToTranscript(new VCRichTextParagraph("command", commandRunList));
}


/**
 * Posts the VCRichTextParagraph to the status viewer.
 * <p>
 * @param commandRunParagraph the rich-text status to post
 */
public void postStatus(VCRichTextParagraph commandRunParagraph) {

        getDesktopStatusViewer().postStatus(commandRunParagraph.toString());

        pasteToTranscript(commandRunParagraph);
}


/**
 * Posts the status.
 * <p>
 * @param commandRunParagraph the rich-text status to post
 */
public void postStatus(DVError error) {

        traceMessage(getClass() + "  >>  postStatus = " + error);

        getDesktopStatusViewer().postErrorStatus(error.toString());

        pasteToTranscript(error);
}


/**
 * Posts the exception to the status viewer.
 * <p>
 * @param exception - DVException
 */
public void postStatus(DVException exception) {

        traceMessage(getClass() + "  >>  postStatus(exception) = " + exception);

        getDesktopStatusViewer().postWarningStatus(exception.toString());

        pasteToTranscript(exception);
}


/**
 * Posts the text to the status viewer.
 * <p>
 * @param statusText the status text to post
 */
public void postStatus(String statusText) {

        getDesktopStatusViewer().postStatus(statusText);
}


/**
 * Posts a warning message to the status viewer.
 */
public void postWarningStatus(String message) {

        getDesktopStatusViewer().postWarningStatus(message);
```

```
        pasteToTranscript(message);
    }


    /**
     * Called when a bound property is changed,
     * <p>
     * @param evt PropertyChangeEvent
     */
    public void propertyChange(PropertyChangeEvent event) {

        tracePropertyChange(event);

        if (event.getPropertyName().equals("SQL Query Available")) {
                Object newValue = event.getNewValue();
                if (newValue instanceof VCRichTextParagraph)
                        postStatus((VCRichTextParagraph)newValue);
                else if (newValue instanceof MDbNodeTreeData)
                        postStatus(((MDbNodeTreeData)newValue).toString());
                else if (newValue instanceof DVException)
                        postStatus((DVException)newValue);
                else if (newValue instanceof DVError)
                        postStatus((DVError)newValue);
        } else if (event.getPropertyName().equals("DB Results Exception"))
                postStatus((DVException)event.getNewValue());
        else if (event.getPropertyName().equals("DB Results Error"))
                postStatus((DVError)event.getNewValue());
        else if (event.getPropertyName().equals("Exception Raised"))
                postStatus((DVException)event.getNewValue());
        else if (event.getPropertyName().equals("Error Raised"))
                postStatus((DVError)event.getNewValue());
        else if
    (event.getPropertyName().equals(IterationControllerFrame.ITERATION_CONTROL_EXCEPTION))
                postStatus((DVException)event.getNewValue());
        else if (event.getPropertyName().equals("Iteration Control Error"))
                postStatus((DVError)event.getNewValue());
        else if
    (event.getPropertyName().equals(TranscriptController.CLEAR_DESKTOP_STATUS))
                clearDesktopStatus();
        else if (event.getPropertyName().equals(TranscriptController.OPERATION_BEGUN))
                postOperationBegun((String)event.getNewValue());
        else if
    (event.getPropertyName().equals(TranscriptController.OPERATION_CANCELED))
                postOperationCanceled((String)event.getNewValue());
        else if (event.getPropertyName().equals(TranscriptController.OPERATION_FAILED))
                postOperationFailed((String)event.getNewValue());
        else if
    (event.getPropertyName().equals(TranscriptController.OPERATION_FINISHED))
                postOperationFinished((String)event.getNewValue());
        else if (event.getPropertyName().equals(TranscriptController.POST_ERROR))
                postErrorStatus((String)event.getNewValue());
        else if (event.getPropertyName().equals(TranscriptController.POST_STATUS))
                postStatus((String)event.getNewValue());
        else if (event.getPropertyName().equals(TranscriptController.POST_WARNING))
                postWarningStatus((String)event.getNewValue());
    }

    /**
     * Saves the active transcript.
     */
    public void saveAction() {

        try {
                getTranscriptTabsViewer().saveTranscript();

                refresh();
        } catch (CannotSaveAPreviousMasterTranscriptException exception) {
                postWarningStatus(exception.toString());
        }
    }

    /**
     * "Save As" the active transcript.
     */
    public void saveAsAction() {
```

```
        try {
                getTranscriptTabsViewer().saveAsTranscript();

                refresh();
        } catch (CannotSaveAsMasterTranscriptException exception) {
                postWarningStatus(exception.toString());
        }
}

/**
 * Sets the host of the frame.
 */
public void setDesktopView(JLayeredPane desktopView) {

        this.desktopView = desktopView;
}


/**
 * Sets the previous executed command.
 * <p>
 * @param commandText - String
 */
public void setPreviousExecutedCommand(String commandText) {

        previousExecutedCommand = commandText;
}
}


package com.dcr.dvg.view.controller.transcript;
/**
 * @(#)TranscriptControllerFrame.java
 * <p>
 * ********************************************************************************
* ********************************************************************************
 * <p>
 * The class <code>TranscriptControllerFrame</code> is the frame view for the
 *   TranscriptControllerFrame.
 * <p>
 * @author              Edward L. Stull
 * @version 1.5
 * @since               JDK 2
 */
//3456789012345678901234567890123456789012345678901234567890123456789*12345678901234567890123456789012345678901234567890

import java.beans.PropertyChangeEvent;

import javax.swing.JLayeredPane;

import com.dcr.dve.view.vcomponent.vctext.IVCEditor;

import com.dcr.dvg.view.component.desktop.DVGDesktop;
import com.dcr.dvg.view.component.text.rich.IRichTextFileEditorEmbedded;
import com.dcr.dvg.view.controller.DesktopControllerFrame;

public class TranscriptControllerFrame extends DesktopControllerFrame {
/**
 * Constructor using the desktop view.
 * <p>
 * The context for the new frame is set by the desktop.
 * <p>
 * @param desktopView - JLayeredPane
 */
public TranscriptControllerFrame(JLayeredPane desktopView) {

        super();

        setDesktopView(desktopView);

        initialize();
}


/**
```

```
 * Gets the (active) transcript editor.
 * <p>
 * @return com.dcr.dve.view.vcomponent.VCEditor
 */
public IVCEditor getEditor() {

        return getTranscriptController().getEditor();
}


/**
 * Gets the Transcript Controller.
 * <p>
 * @return transcriptController - TranscriptController
 */
public TranscriptController getTranscriptController() {

        return (TranscriptController)getViewer();
}


/**
 * Initializes.
 */
public void initialize() {

        setViewer(new TranscriptController(getDesktopView()));

        getTranscriptController().commandsSetEnabled(true);
        addPropertyChangeListener(getTranscriptController());

        setContentPane(getTranscriptController());
        setTitle("Transcript Controller");
}


/**
 * Called when a bound property is changed,
 * <p>
 * @param evt PropertyChangeEvent
 */
public void propertyChange(PropertyChangeEvent event) {

        /*debug trace*///System.out.println(getClass() + ">>propertyChange on \"" +
event.getPropertyName() + "\" from " + event.getSource().getClass());
        if (event.getPropertyName().equals("isClosed"))
                finalizeListenerIsClosed(event.getSource());
        else if (event.getPropertyName().equals(DVGDesktop.LAYER_PROPERTY))
                super.propertyChange(event);
        else
                getTranscriptController().propertyChange(event); // forward to the
transcript viewer
}
}


package com.dcr.dvg.view.controller.transcript;
/**
 * @(#)TranscriptTabsViewer.java
 * <p>
 * ********************************************************************************
 * ********************************************************************************
 * <p>
 * The class <code>TranscriptTabsViewer</code> is the view that contains all
 *   of the loaded transcripts, portrayed in a tab viewer.
 * <p>
 * @author              Edward L. Stull
 * @version 1.4
 * @since               JDK 2
 */
//345678901234567890123456789012345678901234567890123456789*12345678901234567890123456789

import java.awt.Component;
import java.beans.PropertyChangeEvent;
import java.io.File;
import java.io.IOException;
```

```
import java.util.Date;

import javax.swing.JTabbedPane;
import javax.swing.RepaintManager;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

import com.dcr.dve.model.muser.MUserContext;
import com.dcr.dve.view.vcomponent.vctext.VCRichTextEditor;
import com.dcr.dve.view.vcomponent.vctext.IVCTextFileEditor;
import com.dcr.dve.view.vcomponent.vctext.VCRichTextParagraph;
import com.dcr.dve.view.vprocess.IVPView;
import com.dcr.dve.view.vprocess.VPTabsViewer;
//import com.dcr.dve.view.vcomponent.vctext.IVCEditor;
//import com.dcr.dve.view.vcomponent.vctext.IVCTextFileEditor;
import com.dcr.dvg.model.transcript.exception.CannotCloseMasterTranscriptException;
import
com.dcr.dvg.model.transcript.exception.CannotSaveAPreviousMasterTranscriptException;
import com.dcr.dvg.model.transcript.exception.CannotSaveAsMasterTranscriptException;
import com.dcr.dvg.model.transcript.MasterTranscript;
import com.dcr.dvg.model.transcript.Transcript;
import com.dcr.dvg.util.throwable.DVException;
import com.dcr.dvg.view.component.text.rich.IRichTextFileEditorEmbedded;

public class TranscriptTabsViewer extends VPTabsViewer implements IVPView {

        protected transient TranscriptController controller = null;
        protected transient TranscriptViewer masterTranscript = null;

/**
 * Constructor based on the TranscriptController.
 * <p>
 * The context for the new frame is set by the desktop.
 */
public TranscriptTabsViewer(TranscriptController controller) {

        super();

        this.controller = controller;

        initialize();
}


/**
 * Adds a blank transcript.
 * <p>
 * @param tabName - String - name to use on the transcript's tab
 */
public TranscriptViewer addBlankTranscript(String tabName) {

        TranscriptViewer newTranscript = new TranscriptViewer(this, new Transcript());
        addTab(tabName, newTranscript);
        setSelectedComponent(newTranscript);

        return newTranscript;
}

/**
 * Adds a blank transcript.
 * <p>
 * @return transcriptViewer - TranscriptViewer
 */
public TranscriptViewer addMasterTranscriptViewer() {

        TranscriptViewer newTranscriptViewer = new TranscriptViewer(this, new
MasterTranscript());
        addTab("Master", newTranscriptViewer);
        setSelectedComponent(newTranscriptViewer);

        return newTranscriptViewer;
}

/**
 * Close the transcripts.
 */
```

```
public void close() {

        // close smaster transcript
        makeMasterTranscriptAction();
        try {
                getActiveTranscriptViewer().close();
        } catch (IOException exception) {
                // just ignore for now, to late to do anything about it now
                // post message later
        }

        // close remaining transcripts
        for (int i = 1; i < getTranscriptCount(); i++) {
                setSelectedIndex(i);
                try {
                        if (! isAMasterTranscript())
                                getActiveTranscriptViewer().close(); //"save as" for any
new transcript viewers
                } catch (IOException exception) {
                // just ignore for now, to late to do anything about it now
                // post message later
                }
        }

        revalidate();
}

/**
 * Closes the active transcript.
 */
public void closeTranscript()
  throws CannotCloseMasterTranscriptException, IOException {

        if (! isCurrentMasterTranscript()) {
                getActiveTranscriptViewer().close();

                Component tabComponent = getSelectedComponent();
                remove(tabComponent);

                refresh();
        } else {
                CannotCloseMasterTranscriptException exception = new
CannotCloseMasterTranscriptException();
                throw exception;
        }
}

/**
 * Creates the master transcript and viewer.
 *
 * Assumes that the DVG home path exists with the proper permissions
 */
public void createMasterTranscript() {

        String pathName = MUserContext.getDVGHomePath();
        String fileName = MUserContext.getUserId() + (new
Date()).toString().replace(':', '-');

        File masterTranscriptFile = null;

        try {
                new File(pathName, fileName).createNewFile();
                masterTranscriptFile = new File(pathName, fileName);
        } catch (IOException exception) {
                fileName = null;
                System.out.println("Error upon creating the Master Transcript
file>>:\n" + exception);
        }

        masterTranscript = addMasterTranscriptViewer();

        refresh();

        VCRichTextEditor editor =
((VCRichTextEditor)masterTranscript.getTranscriptEditor());
        editor.setEditable(false);
```

```
        editor.setFileBeingEdited(masterTranscriptFile);
    }

    /**
     * Gets the active transcript viewer.
     * <p>
     * @return transcriptViewer  TranscriptViewer
     */
    public TranscriptViewer getActiveTranscriptViewer() {

        traceMessage(getClass() + " getActiveTranscript()=" +
    getTitleAt(getSelectedIndex()));

        return (TranscriptViewer)getSelectedComponent();
    }


    /**
     * Gets the controller for this component.
     * <p>
     * @return controller - TranscriptController
     */
    public TranscriptController getController() {

        return controller;
    }


    /**
     * Gets the transcript's editor.
     * <p>
     * @return editor IRichTextFileEditorEmbedded
     */
    public IRichTextFileEditorEmbedded getEditor() {

        return getActiveTranscriptViewer().getTranscriptEditor();
    }


    /**
     * This a standard interface to access the component's editor.
     * <p>
     * Gets the Master Transcript (viewer).
     * <p>
     * @return masterTranscript - TranscriptViewer
     */
    public TranscriptViewer getMasterTranscript() {

        return masterTranscript;
    }


    /**
     * Gets the Transcript at the specified index in the tab viewer.
     * <p>
     * @return masterTranscript - TranscriptViewer
     */
    public TranscriptViewer getTranscriptAt(int index) {

        traceMessage(getClass() + " getTranscriptAt()=" + getTitleAt(index));

        return (TranscriptViewer)getComponentAt(index);
    }


    /**
     * Gets the transcript count.
     * <p>
     * @return transcriptCount - int
     */
    public int getTranscriptCount() {

        return getTabCount();
    }
```

```
/**
 * Initializes this viewer.
 */
public void initialize() {

        instance = this; // for inner class reference

        createMasterTranscript();
}

/**
 * Answer true if the active transcript is the master transcript.
 * <p>
 * @return mode boolean
 */
public boolean isAMasterTranscript() {

        return getActiveTranscriptViewer().isAMasterTranscript();
}

/**
 * Answer true if the active transcript is the current (tab 0) master transcript.
 * <p>
 * @return mode boolean
 */
public boolean isCurrentMasterTranscript() {

        return getSelectedIndex() == 0;
}

/**
 * Answer true if the active transcript is unnamed.
 * <p>
 * @return mode boolean
 */
public boolean isCurrentTranscriptUnnamed() {

        return getActiveTranscriptViewer().getFileBeingEdited() == null;
}

/**
 * Makes the Master Transcript the active transcript.
 */
public void makeMasterTranscriptAction() {

        setSelectedIndex(0);
}

/**
 * Creates and opens a "new" Transcript.
 */
public void newTranscript() {

        newTranscript(">> NEW <<");
}

/**
 * Creates and opens a "new" Transcript with the specified name.
 *
 * @param tabName - String - name of the new transcript
 */
public TranscriptViewer newTranscript(String tabName) {

        TranscriptViewer newTranscript = addBlankTranscript(tabName);

        refresh();

        return newTranscript;
}

/**
 * Opens and makes active an existing transcript from a file.
 * Can open only in a new "user" transcript.
 */
public void openTranscript() {
```

```
        File file = getActiveTranscriptViewer().getFileToOpen();

        if (file != null) {
                TranscriptViewer newTranscript = addBlankTranscript(file.getName());
                getActiveTranscriptViewer().openTranscript(file);
        }

        refresh();
}

/**
 * Pastes rich text to the transcripts.
 * <p>
 * @param commandRunParagraph - VCRichTextParagraph
 */
public void pasteRichTextToTranscript(VCRichTextParagraph commandRunParagraph) {

        for (int i=0; i < getTranscriptCount(); i++)
                if (isCurrentMasterTranscript() || (! isAMasterTranscript()))

        ((VCRichTextEditor)getTranscriptAt(i).getTranscriptEditor()).appendParagraph(co
mmandRunParagraph);
}

/**
 * Called when a bound property is changed,
 * <p>
 * @param evt PropertyChangeEvent
 */
public void propertyChange(PropertyChangeEvent event) {

        tracePropertyChange(event);

        if (event.getPropertyName().equals("Exception Raised"))
                getController().propertyChange(event);
        else if (event.getPropertyName().equals("Error Raised"))
                getController().propertyChange(event);
}


/**
 * "Save As" the active transcript.
 */
public void saveAsTranscript()
  throws CannotSaveAsMasterTranscriptException {

        getActiveTranscriptViewer().saveAsTranscript();

        setCurrentTranscriptTabTitle();
}

/**
 * Saves the active transcript.
 */
public void saveTranscript()
  throws CannotSaveAPreviousMasterTranscriptException {

        if ((! isAMasterTranscript()) || isCurrentMasterTranscript()) {
                getActiveTranscriptViewer().saveTranscript();
        } else {
                CannotSaveAPreviousMasterTranscriptException exception = new
CannotSaveAPreviousMasterTranscriptException();
                throw exception;
        }
}

/**
 * Sets the title of the current transcript .
 */
public void setCurrentTranscriptTabTitle() {

        File file = getActiveTranscriptViewer().getFileBeingEdited();

        if (file != null) {
                setTitleAt(getSelectedIndex(), file.getName());
                refresh();
```

```
        }
    }
    }


    package com.dcr.dvg.view.controller.transcript;
    /**
     * @(#)TranscriptViewer.java
     * <p>
     * ********************************************************************************
     * ********************************************************************************
     * <p>
     * The class <code>TranscriptViewer</code> is the view of a loaded transcript.
     * <p>
     * @author              Edward L. Stull
     *      @version 1.3
     * @since               JDK1.1
     */
    //345678901234567890123456789012345678901234567890123456789*12345678901234567890123456789

    import java.awt.Component;
    import java.beans.PropertyChangeListener;
    import java.beans.PropertyChangeEvent;
    import java.io.IOException;
    import java.io.File;

    import com.dcr.dve.view.vcomponent.vcpanel.VCScrollViewer;
    import com.dcr.dve.view.vcomponent.vctext.IVCTextFileEditor;
    import com.dcr.dve.view.vcomponent.vctext.VCRichTextEditor;
    import com.dcr.dve.view.vprocess.VPRichTextEditor;

    import com.dcr.dvg.model.transcript.exception.CannotSaveAsMasterTranscriptException;
    import com.dcr.dvg.model.transcript.MasterTranscript;
    import com.dcr.dvg.model.transcript.Transcript;
    import com.dcr.dvg.view.component.text.ITextFileEditorEmbedded;
    import com.dcr.dvg.view.component.text.rich.IRichTextFileEditorEmbedded;

    public class TranscriptViewer
      extends VCScrollViewer
      implements PropertyChangeListener {

            protected TranscriptTabsViewer controller = null;
            protected VPRichTextEditor editor = null;

    /**
     * Constructor.
     * <p>
     * @param controller - TranscriptTabsViewer
     * @param transcript - Transcript
     */
    public TranscriptViewer(TranscriptTabsViewer controller, Transcript transcript) {

            this.controller = controller;

            editor = new VPRichTextEditor(transcript); // initial document
            editor.addPropertyChangeListener(this);
            editor.createDefaultStyles();

            setViewportView(editor);
            setBackingStoreEnabled(true);
    }


    /**
     * Closes the transcript viewer and its transcript (file).
     */
    public void close()
      throws IOException {

            closeTranscript();
    }

    /**
     * Closes a transcript.
     */
    public void closeTranscript()
```

```
 throws IOException {

        getTranscriptEditor().saveFile();

        getTranscriptEditor().closeAction();
}

/**
 * Gets the controller for this component.
 * <p>
 * @return controller - TranscriptTabsViewer
 */
public TranscriptTabsViewer getController() {

        return controller;
}


/**
 * Gets the transcript file being edited.
 * <p>
 * @return file - File
 */
public File getFileBeingEdited() {

        return getTranscriptEditor().getFileBeingEdited();
}

/**
 * Gets a transcript file to open.
 * <p>
 * @return file - File
 */
public File getFileToOpen() {

        return getTranscriptEditor().getFileToOpen();
}

/**
 * Gets the transcript's editor.
 * <p>
 * @return editor - IRichTextFileEditorEmbedded
 */
public IRichTextFileEditorEmbedded getTranscriptEditor() {

        return (IRichTextFileEditorEmbedded)editor;
}


/**
 * Answer true if the active transcript is the master transcript.
 * <p>
 * @return mode boolean
 */
public boolean isAMasterTranscript() {

        return getTranscriptEditor().getDocument() instanceof MasterTranscript;
}

/**
 * Opens an existing transcript file.
 */
public void openTranscript(File file) {

        getTranscriptEditor().open(file);
}

/**
 * Called when a bound property is changed,
 * <p>
 * @param evt PropertyChangeEvent
 */
public void propertyChange(PropertyChangeEvent event) {

        if (TRACE)
                tracePropertyChange(event);
```

```
        if (event.getPropertyName().equals("Exception Raised"))
                getController().propertyChange(event);
        else if (event.getPropertyName().equals("Error Raised"))
                getController().propertyChange(event);
}


/**
 * "Save As" the transcript.
 */
public void saveAsTranscript()
  throws CannotSaveAsMasterTranscriptException {

        if (! isAMasterTranscript()) {
                getTranscriptEditor().saveAsAction();
        } else {
                CannotSaveAsMasterTranscriptException exception = new
CannotSaveAsMasterTranscriptException();
                throw exception;
        }
}

/**
 * Saves the active transcript to its exiting file.
 */
public void saveTranscript() {

        getTranscriptEditor().saveAction();
}
}
```

Right-click for Help

```
package com.dcr.dve.view.vcomponent.vcmenu;
/**
 * @(#)IVCMenuItem.java
 * <p>
 * ********************************************************************************
 * ********************************************************************************
 * <p>
 * IVCMenuItem provides a common interface for VCMenuItem components.
 * <p>
 * @author              Edward L. Stull
 *      @version 1.1
 * @since               JDK 2
 */
//3456789012345678901234567890123456789012345678901234567890123456789*123456789012345678901234567890

import java.awt.event.ActionListener;
import java.awt.event.MouseEvent;

import javax.swing.Icon;
import javax.swing.MenuElement;
import javax.swing.MenuSelectionManager;

import com.dcr.dve.view.vcomponent.vccontrol.IVCCHelp;

public interface IVCMenuItem extends IVCCHelp {
/**
 * Adds an Action Listener
 * <p>
 * @param listener - ActionListener
 */
public void addActionListener(ActionListener listener);


/**
 * Initializes the menu item and register the result with the menuitem
 *   hashtable so that it can be fetched with getMenuItem().
 * <p>
 * @see #getMenuItem
 */
public void initialize();


/**
```

```
 * Overrides the processing of the event modifiers.
 * <p>
 * Process a mouse event. event is a MouseEvent with source being the receiving
component.
 *   componentPath is the path of the receiving MenuElement in the menu
 *   hierarchy. manager is the MenuSelectionManager for the menu hierarchy.
 * This method should process the MouseEvent and change the menu selection if
necessary
 *   by using MenuSelectionManager's API.
 * <p>
 * Note: you do not have to forward the event to sub-components. This is done
automatically
 *   by the MenuSelectionManager
 */
public void processMouseEvent(MouseEvent event, MenuElement path[],
MenuSelectionManager manager);


/**
 * Sets the Action Command
 * <p>
 * @param commandName java.lang.String
 */
public void setActionCommand(String commandName);


/**
 * Sets the enable mode.
 * <p>
 * @param mode boolean
 */
public void setEnabled(boolean mode);


/**
 * Sets the horizontal text position
 * <p>
 * @param alignment - int
 */
public void setHorizontalTextPosition(int alignment);


/**
 * Sets the icon
 * <p>
 * @param icon - Icon
 */
public void setIcon(Icon icon);


/**
 * Sets the mnemonic.
 * <p>
 * @param keyAccelerator - int (char)
 */
public void setMnemonic(int keyAccelerator);


/**
 * Sets the text.
 * <p>
 * @param text java.lang.String
 */
public void setText(String text);
}


package com.dcr.dve.view.vcomponent.vccontrol;
/**
 * @(#)IVCCHelp.java
 * <p>
 * ********************************************************************************
 * ********************************************************************************
 * <p>
 * IVCCHelp provides a common interface for help-related components.
 * <p>
```

```
 * @author             Edward L. Stull (ID: ELS)
 *      @version 2.006
 * @since              JDK 2
 */
//3456789012345678901234567890123456789012345678901234567890123456789*123456789012345678901234567890

import java.awt.event.MouseEvent;

import javax.swing.MenuElement;
import javax.swing.MenuSelectionManager;

public interface IVCCHelp {
/**
 * Gets the name of the command.
 * <p>
 * @return name java.lang.String
 */
public String getCommandName();


/**
 * Gets the default name of the command.
 * <p>
 * @return name java.lang.String
 */
public String getDefaultName();


/**
 * Gets the string associated with the resource tag.
 * <p>
 * @param name java.lang.String
 * @return name java.lang.String
 */
public String getResourceString(String resourceTagName);
}


package com.dcr.dve.view.vcomponent.vccontrol;
/**
 * @(#)VCControlKit.java
 * <p>
* ********************************************************************************
 * ********************************************************************************
 * <p>
 * An <code>VCControlBar</code> is a set of helper methods supporting
 *  the operation of controls, that is, active components. Currently, this
 *  kit provides context help support in menu and toolbar controls.
 * <p>
 * @author             Edward L. Stull
 * @version 2.11
 * @since              JDK 2
 */
//3456789012345678901234567890123456789012345678901234567890123456789*123456789012345678901234567890

import java.awt.event.MouseEvent;

import javax.swing.MenuElement;
import javax.swing.MenuSelectionManager;

import com.dcr.dve.view.vcomponent.vchtml.VCDefaultBrowser;

public class VCControlKit extends Object {
/**
 * Gets the URL text for the contextual help name.
 * <p>
 * @param helpTag - a resource tag for help
 * @return contextHelpUrlName
 */
public static String getContextHelpUrlLabelFor(String helpTag) {

        return helpTag + getContextHelpUrlLabelSuffix();
}


/**
```

```
 * Gets the label for the contextual help URL.
 * <p>
 * @return contextHelpUrlLabel
 */
public static String getContextHelpUrlLabelSuffix() {

	return "HelpUrl";
}


/**
 * Gets the URL text for the contextual help name using a view that implements
 *   the help interface.
 * <p>
 * @param helpview - a view that implements the help interface
 * @return contextHelpUrlText
 */
public static String getContextHelpUrlTextFor(IVCCHelp helpView) {

	return helpView.getResourceString(helpView.getCommandName() +
getContextHelpUrlLabelSuffix());
}


/**
 * Answer if this is a help context event.
 * <p>
 * @param event - the mouse event.
 * @return boolean
 * @see		java.awt.event.MouseEvent
 * @see		java.awt.event.MouseListener
 * @see		java.awt.Component#addMouseListener
 * @see		java.awt.Component#enableEvents
 */
public static boolean isContextHelpEvent(MouseEvent event) {

	return ((event.getID() == MouseEvent.MOUSE_RELEASED)
			&& ((event.getModifiers() & MouseEvent.META_MASK) != 0));
}


/**
 * Launch the help facility for a view that implements
 *   the help interface.
 * <p>
 * @param helpview - a view that implements the help interface
 */
public static void launchViewerContextHelpUsing(IVCCHelp helpView) {

	String defaultHelpUrlText = getContextHelpUrlTextFor(helpView);

	String fileSep = System.getProperty("file.separator"); // e.g., "/"

	String helpUrlPath = System.getProperty("user.dir") + fileSep + "Help" +
fileSep;

	VCDefaultBrowser browser = new VCDefaultBrowser();
	if (defaultHelpUrlText == null)
		browser.displayURL(helpUrlPath + "Action" + fileSep +
helpView.getDefaultName().replace(' ', '_') + ".htm");
	else
		browser.displayURL(helpUrlPath + defaultHelpUrlText);
}

/**
 * Launch the help facility for this viewer's context using a help URL text.
 * <p>
 * @param helpUrlText - String
 */
public static void launchViewerContextHelpUsing(String helpUrlText) {

	if (helpUrlText == null)
		helpUrlText = "index.htm";

	String fileSep = System.getProperty("file.separator"); // e.g., "/"
```

```
        VCDefaultBrowser browser = new VCDefaultBrowser();
        browser.displayURL(System.getProperty("user.dir") + fileSep + "Help" + fileSep
+ helpUrlText);
}
}


package com.dcr.dve.view.vcomponent.vctool;
/**
 * @(#)VCToolBarButton.java
 * <p>
 * ********************************************************************************
 * ********************************************************************************
 * <p>
 * An implementation of a button for the viewer toolbars.
 * <p>
 * @author              Edward L. Stull
 *      @version 2.009
 * @since               JDK 2
 */
//345678901234567890123456789012345678901234567890123456789+12345678901234567890123456789

import java.awt.Insets;
import java.awt.event.MouseEvent;
import java.net.URL;
import java.util.Hashtable;
import java.util.ResourceBundle;

import javax.swing.ImageIcon;
import javax.swing.border.Border;
import javax.swing.border.CompoundBorder;
import javax.swing.border.EmptyBorder;
import javax.swing.border.EtchedBorder;

import com.dcr.dve.view.vcomponent.VCAction;
import com.dcr.dve.view.vcomponent.VCPushButton;
import com.dcr.dve.view.vcomponent.vcbutton.VCButton;
import com.dcr.dve.view.vcomponent.vccontrol.IVCCHelp;
import com.dcr.dve.view.vcomponent.vccontrol.VCControlKit;
import com.dcr.dve.view.vcomponent.vchtml.VCDefaultBrowser;

import com.dcr.dvg.model.ResourcesKit;

public class VCToolBarButton extends VCPushButton implements IVCCHelp {

        protected VCToolBar controller = null; // the controller for this component
        protected String defaultName;
        protected String tagName; // name used for tag in the resource
        protected ImageIcon imageIcon;

        protected final static Border emptyBorder = new EmptyBorder(2, 2, 2, 2);
        protected final static Border etchedBorder
          = new CompoundBorder(new EtchedBorder(), emptyBorder);

/*
 * Constructor.
 * <p>
 * @param controller - the tool bar controller
 * @param tagName name used for tag in the resource
 */
public VCToolBarButton(VCToolBar controller, String tagName) {

        super();

        initialize(controller, tagName);
}


/**
 * Launchs the help facility for this viewer's context.
 */
public void contextHelpAction() {

        VCControlKit.launchViewerContextHelpUsing(this);
}
```

```
/**
 * Gets the action for the specified command.
 * <p>
 * @param      String - command
 */
protected VCAction getAction(String command) {

        return (VCAction)getCommands().get(command);
}


/**
 * Gets the action suffix string.
 * <p>
 * @return image - suffix
 */
public String getActionSuffix() {

        return getController().getActionSuffix();
}


/**
 * Gets Y alignment.
 * <p>
 * @return alignment
 */
public float getAlignmentY() {

        return 0.5f;
}


/**
 * Gets the action name for the menu item as specified in the resource.
 * <p>
 * @param      String- command
 */
protected VCAction getButtonAction(String commandName) {

        String actionName = getResourceString(commandName + getActionSuffix());

        if (actionName == null)
                actionName = commandName; //defalut action name to that of the command
name

        return getAction(actionName);
}


/**
 * Gets the name (that is, tag name) for the buttom that is used by the resource.
 * <p>
 * @return tagName - java.lang.String
 */
public String getCommandName() {

        return tagName;
}


/**
 * Gets the commands of the controller.
 * <p>
 * @return  commands - java.util.Hashtable;
 */
public Hashtable getCommands() {

        return getController().getCommands();
}


/**
 * Gets the controller for this resource-employing component.
 * <p>
 * @return controller - the VCToolBar controller
```

```
 */
public VCToolBar getController() {

        return controller;
}


/**
 * Gets the name for the buttom that is used by the resource.
 * <p>
 * @return defaultName - String
 */
public String getDefaultName() {

        return defaultName;
}


/**
 * Gets the image(icon) based on the image name.
 * <p>
 * @param imageName java.lang.String
 * @return ImageIcon
 */
public ImageIcon getImageIcon(String imageName) {

        return getController().getImageIcon(imageName);
}


/**
 * Gets the image suffix string.
 * <p>
 * @return imageSuffix - String
 */
public String getImageSuffix() {

        return getController().getImageSuffix();
}


/**
 * Gets the label suffix string.
 * <p>
 * @return labelSuffix - String
 */
public String getLabelSuffix() {

        return getController().getLabelSuffix();
}


/**
 * Gets a resource from the local resource or from a super class resource.
 * <p>
 * @param key
 * @return URL
 */
public URL getResource(String key) {

        return getController().getResource(key);
}


/**
 * Gets the resources.
 * <p>
 * @return ResourceBundle
 */
public ResourceBundle getResources() {

        return getController().getResources();
}


/**
```

```
 * Gets a resource string from the local resource or from a super class resource.
 * <p>
 * @param key java.lang.String
 * @return java.lang.String
 */
public String getResourceString(String key) {

        return getController().getResourceString(key);
}


/**
 * Gets the image suffix string.
 * <p>
 * @return tipSuffix - String
 */
public String getTipSuffix() {

        return getController().getTipSuffix();
}


/**
 * Initializes the menu item and register the result with the menuitem
 *   hashtable so that it can be fetched with getMenuItem().
 * <p>
 * @param controller
 * @param tagName name used for tag in the resource
 * @see #getMenuItem
 */
public void initialize(VCToolBar controller, String tagName) {

        setController(controller);

        setUI();
        setCommandName(tagName);
        setLabelFromResource();
        setCrossActionAndButtonRegistration();
        setImageFromResource();
        setToolTipText();
}


/**
 * Answer if the icon is to be shown.
 */
public boolean isShowIcon() {

        return true;
}


/**
 * Override to process the event modifiers.
 * <p>
 * Processes mouse events occurring on this component by
 * dispatching them to any registered
 * <code>MouseListener</code> objects.
 * <p>
 * This method is not called unless mouse events are
 * enabled for this component. Mouse events are enabled
 * when one of the following occurs:
 * <p><ul>
 * <li>A <code>MouseListener</code> object is registered
 * via <code>addMouseListener</code>.
 * <li>Mouse events are enabled via <code>enableEvents</code>.
 * </ul>
 * @param       event - the mouse event.
 * @see         java.awt.event.MouseEvent
 * @see         java.awt.event.MouseListener
 * @see         java.awt.Component#addMouseListener
 * @see         java.awt.Component#enableEvents
 * @since       JDK1.1
 */
public void processMouseEvent(MouseEvent event) {
```

```
        if (VCControlKit.isContextHelpEvent(event))
                VCControlKit.launchViewerContextHelpUsing(this);
                //((MouseListener)mouseListener.a).model.setPressed(false);
                //mouseListener.model.setPressed(false);
        else
                super.processMouseEvent(event);
}


/**
 * Sets the name for the buttom that is used by the resource.
 * <p>
 * @param tagName
 */
public void setCommandName(String tagName) {

        this.tagName = tagName;
}


/**
 * Sets the controller for this resource-employing component.
 * <p>
 * @param controller - the VCToolBar controller
 */
public void setController(VCToolBar controller) {

        this.controller = controller;
}


/**
 * Cross registers the action and menu item. Also, map
 *   the command name to this menu item.
 */
public void setCrossActionAndButtonRegistration() {

        String commandName = getCommandName();

        VCAction action = getButtonAction(commandName);

        if (action == null)
                action = getButtonAction(getDefaultName());

        if (action != null) {
                // set the action commnand name that is included in the event sent to
action listeners
                setActionCommand((String)action.getValue(action.NAME));
                addActionListener(action);
                setEnabled(action.isEnabled());
                setIcon(action.getIcon());

                action.addPropertyChangeListener(new VCToolBarButtonListener(this));
        } else {
                setEnabled(false);
                /*debug trace*///System.err.println("Action for button \"" +
commandName + "\" not resolved.");
                ResourcesKit.fireResourceTagError(getResources(), getDefaultName());
        }
}


/**
 * Sets the default name for the buttom.
 * <p>
 * @param defaultName
 */
public void setDefaultName(String defaultName) {

        this.defaultName = defaultName;
}


/**
 * Sets the button's image.
 */
```

```
public void setImageFromResource() {

	if (getIcon() == null) { // if not specified by the action
		String commandName = getCommandName();
		URL url = getResource(commandName + getImageSuffix());
		if (url != null) {
			if (isShowIcon())
				setHorizontalTextPosition(VCButton.CENTER);
			else
				setHorizontalTextPosition(VCButton.LEFT);

			setIcon(new
ImageIcon(java.awt.Toolkit.getDefaultToolkit().getImage(url)));
		} else

			// default to image with the command name
			setIcon(getImageIcon(getDefaultName()));
	}
}


/**
 * Sets the menu item's label in the menu list.
 * <p>
 * @param command name
 */
protected void setLabelFromResource() {

	String commandName = getCommandName();
	String label = getResourceString(commandName + getLabelSuffix());

	//defalut the label
	if (label == null)
		label = getResourceString(commandName + getActionSuffix());
	if (label == null)
		label = commandName;

	//setText(label);

	setDefaultName(label);
}


/**
 * Answer if the icon is to be shown.
 */
public boolean setShowIcon() {

	return true;
}


/**
 * For tool bar buttons, override text set to only set the name
 * and the tool tip.
 * <p>
 * @param buttonText - String - the text to display with the button
 */
public void setText(String buttonText) {

	setDefaultName(buttonText);
	setToolTipText(buttonText);
}


/**
 * Sets the tool tip text based on the resource specification.
 */
protected void setToolTipText() {

	String tip = getResourceString(getCommandName() + getTipSuffix());
	if (tip != null)
		super.setToolTipText(tip);
	else
		super.setToolTipText(getDefaultName());
}
```

```
/**
 * Sets UI for this component.
 */
public void setUI() {

        setMargin(new Insets(1,1,1,1));

        Border emptyBorder = new EmptyBorder(1,1,1,1);
        setBorder(emptyBorder);

        setRequestFocusEnabled(false);
}
}
```

United States Patent & Trademark Office

Office of Initial Patent Examination -- Scanning Division

Application deficiencies found during scanning:

☐ Page(s)__________ of ______________________________ were not present for scanning.
(Document title)

☐ Page(s)__________ of ______________________________ were not present for scanning.
(Document title)

☑ ***Scanned copy is best available.*** Drawings are too dark